(12) UK Patent Application (19) GB (11) 2 200 481 (13) A

(43) Application published 3 Aug 1988

(21) Application No 8729324

(22) Date of filing 16 Dec 1987

(30) Priority data
(31) 006015 (32) 22 Jan 1987 (33) US

(71) Applicant
National Semiconductor Corporation

(Incorporated in USA-Delaware)

2900 Semiconductor Drive, Santa Clara,
California 95052-8090,
United States of America

(72) Inventors
Alon Schacham
Jonathan Levy

(74) Agent and/or Address for Service
Lloyd Wise Tregear & Co
Norman House, 105-109 Strand, London, WC2R 0AE

(51) INT CL⁴
G06F 12/12

(52) Domestic classification (Edition J):
G4A MC

(56) Documents cited
EP A2 0227319 WO A1 86/00440

(58) Field of search
G4A
Selected US specifications from IPC sub-class
G06F

# (54) Maintaining coherence between a microprocessor's integrated cache and external memory

(57) A method of maintaining data coherency between a microprocessor's integrated (on-chip) cache memory 14, 16 and its associated external main memory is provided. When data is written to external memory, the address of the external memory write is compared with the address tags of the integrated cache memory entries. If the comparison results in a match, data at locations within the cache corresponding to the write address are invalidated by execution of an invalidation instruction without adversely affecting the microprocessor's performance.

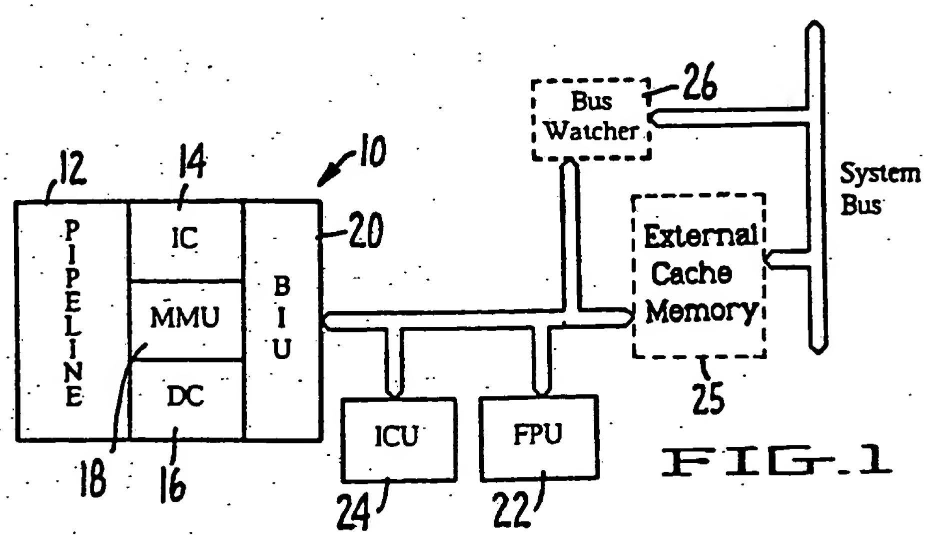


FIG. 1

GB 2 200 481 A

1/15

2200481

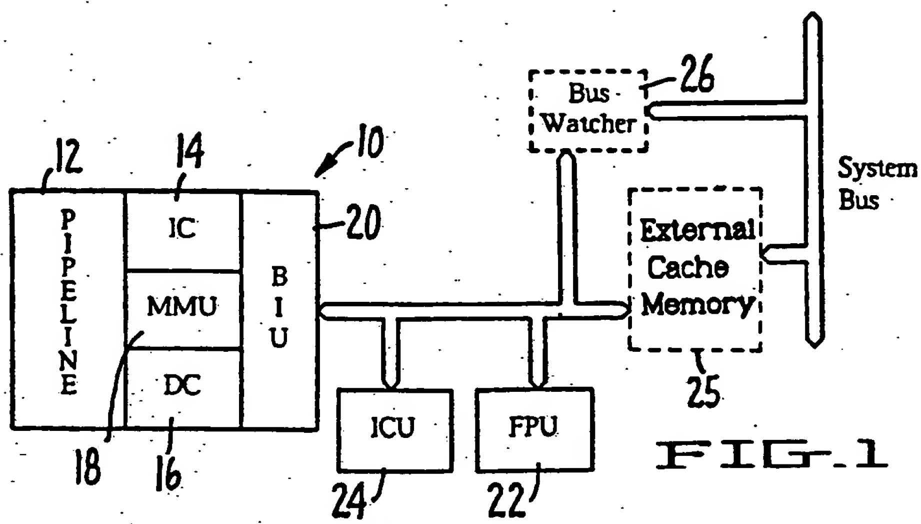


FIG_1

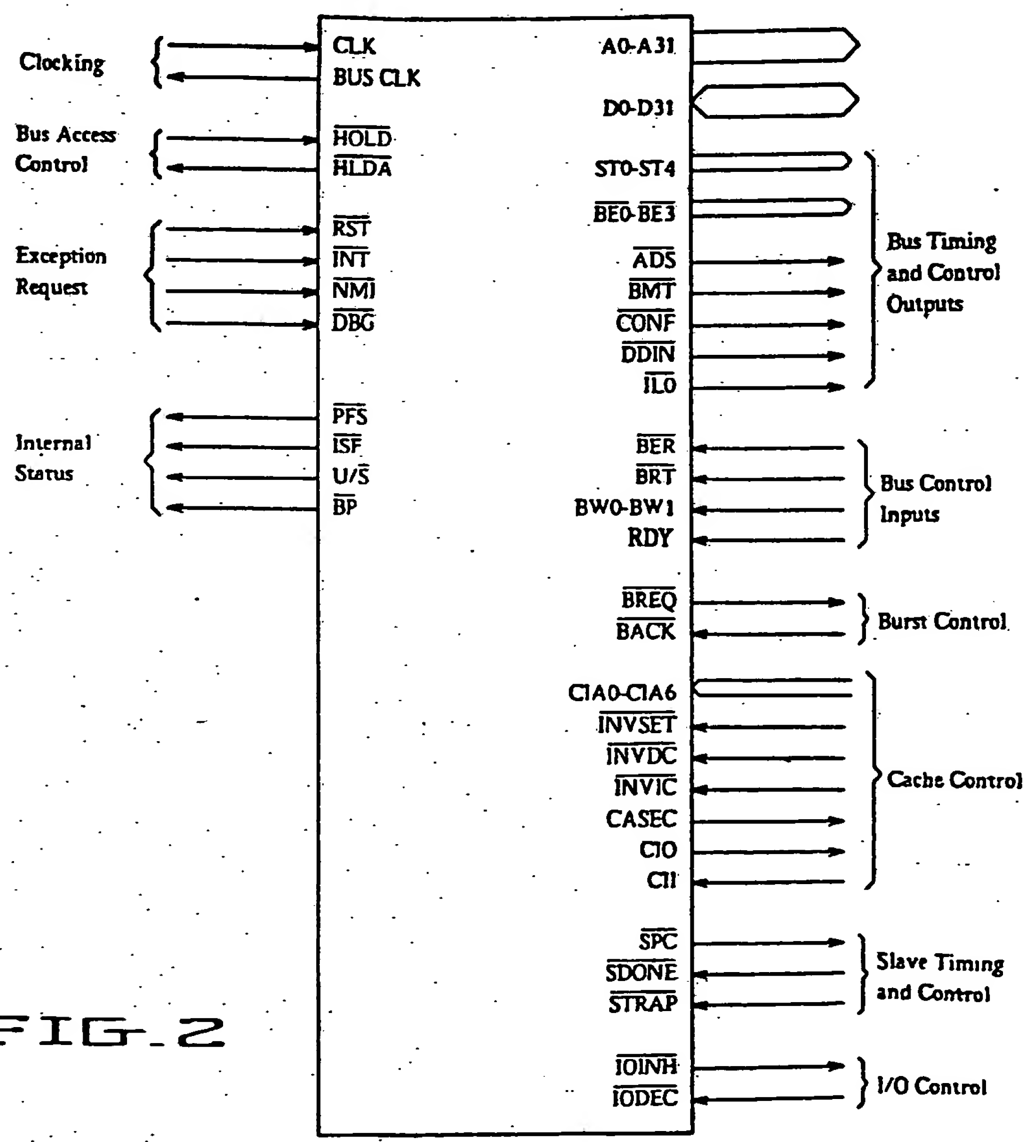


FIG_2

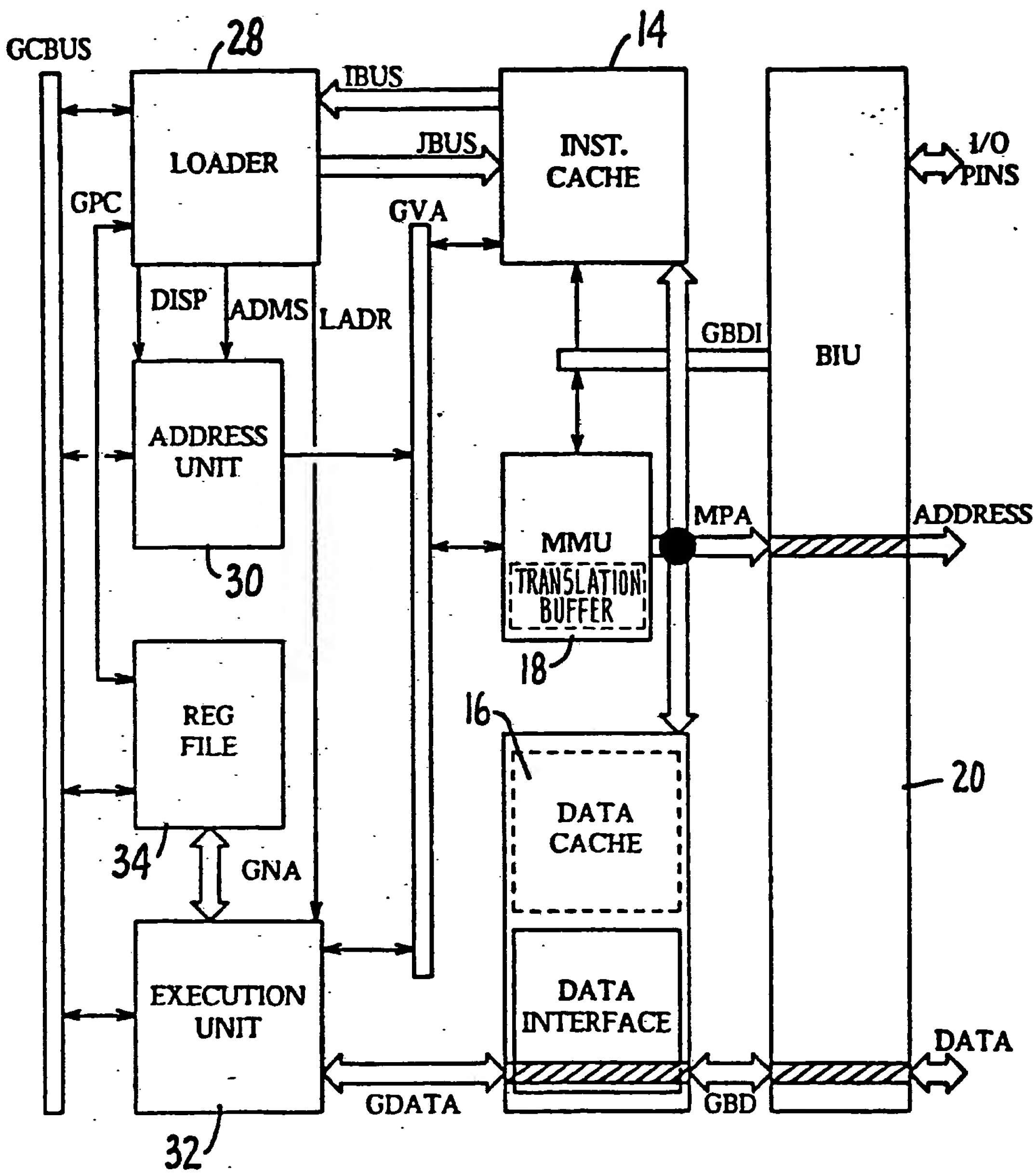
GCBUS
28
LOADER
IBUS
JBUS
14
INST. CACHE
I/O PINS
GPC
GVA
DISP
ADMS
LADR
GBDI
BIU
ADDRESS UNIT
30
MMU
TRANSLATION BUFFER
MPA
ADDRESS
18
16
REG FILE
34
GNA
DATA CACHE
20
EXECUTION UNIT
DATA INTERFACE
DATA
GDATA
GBD
32

FIG. 3

2200481

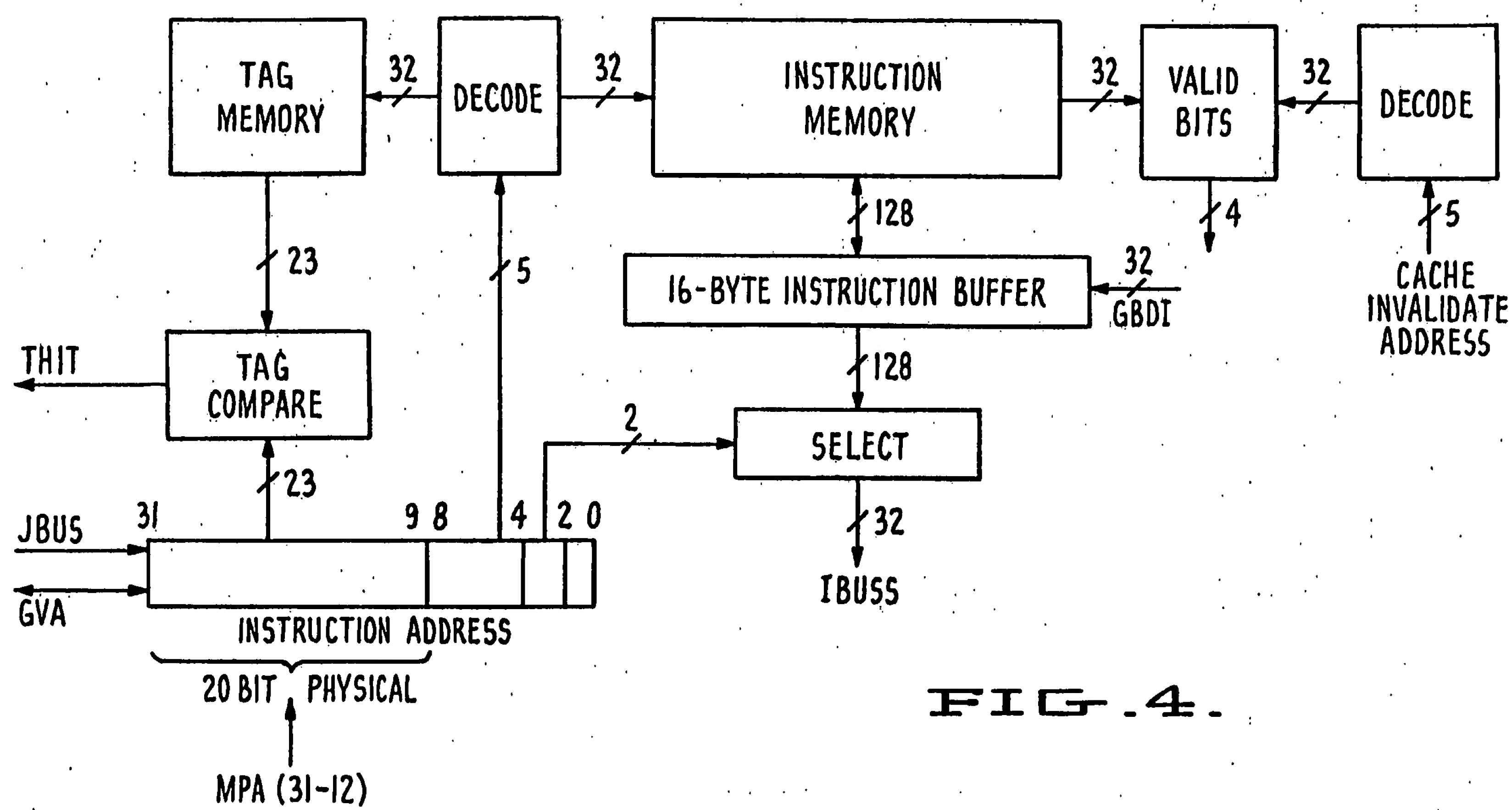


FIG. 4.

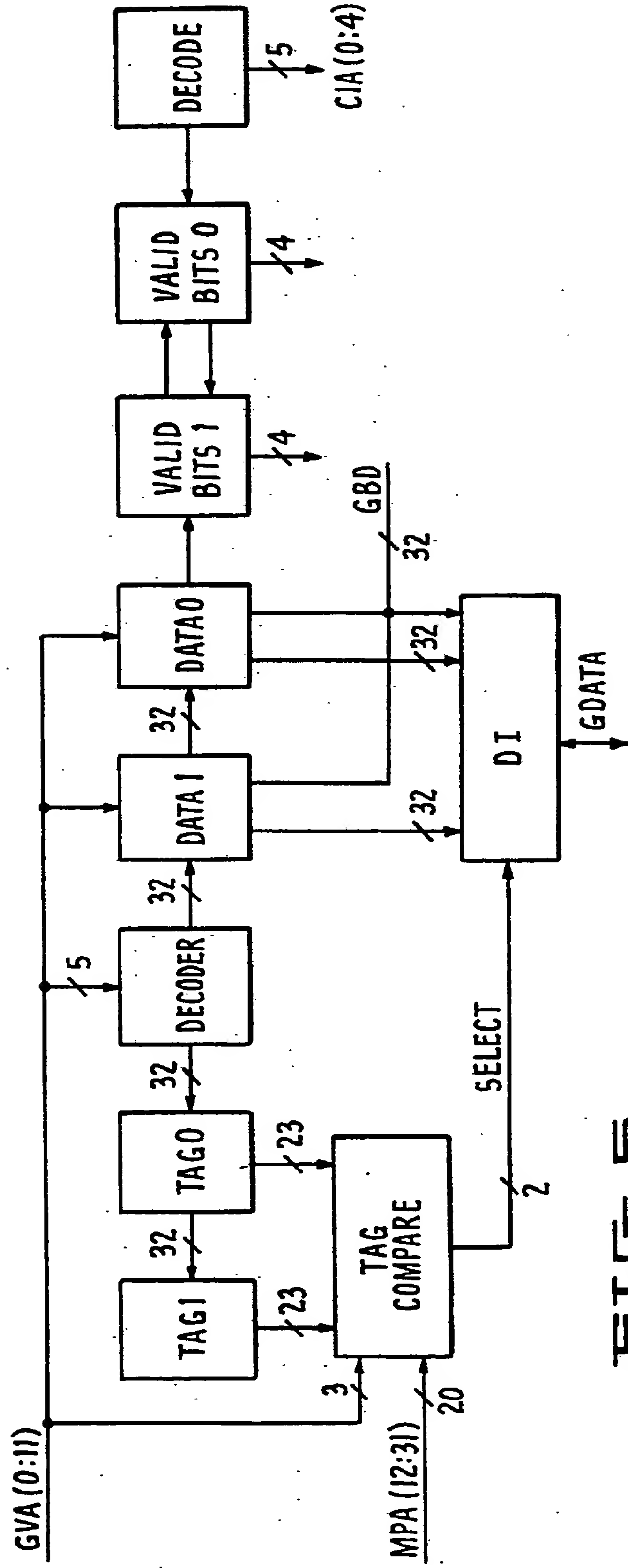
DECODE
5
CIA(0:4)
VALID
BITS 0
4
VALID
BITS 1
4
GBD
32
DATA 0
32
DATA 1
32
DI
GDATA
32
32
DECODER
5
32
TAG 0
23
TAG 1
23
TAG
COMPARE
2
SELECT
32
3
20
GVA(0:11)
MPA(12:31)

FIG. 5.

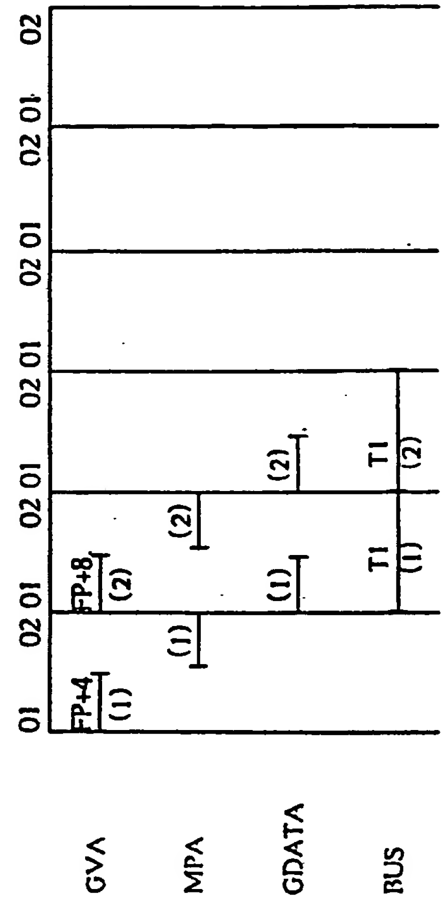
01
02 01
02 01
02 01
02 01
02 01
02 01
02
FP+4
(1)
FP+8
(2)
(1)
(2)
(1)
(2)
T1
(1)
T1
(2)
GVA
MPA
GDATA
BUS

FIG. 6.

2200481

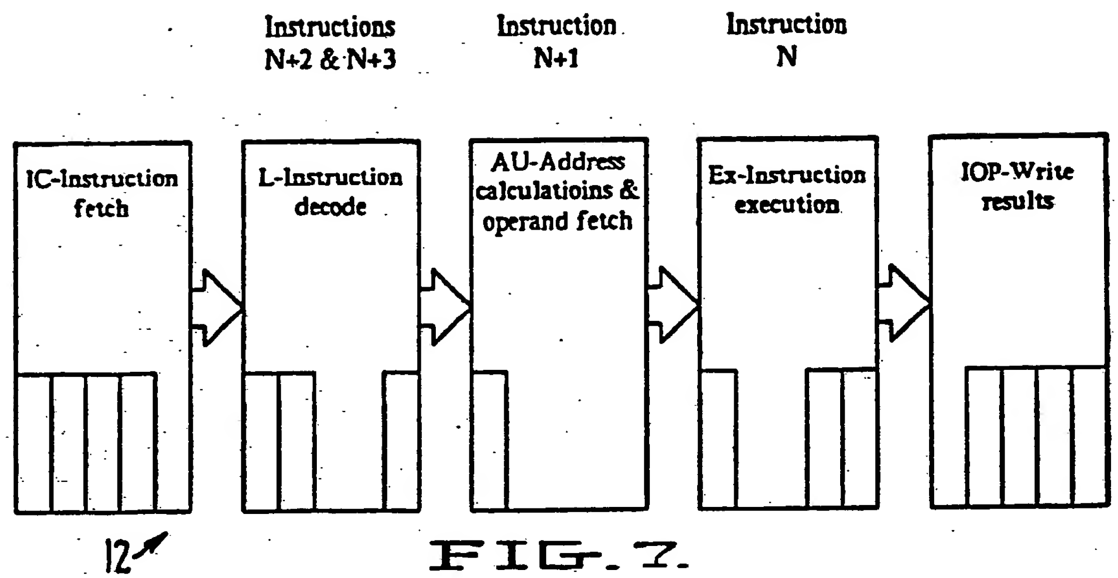


FIG. 7.

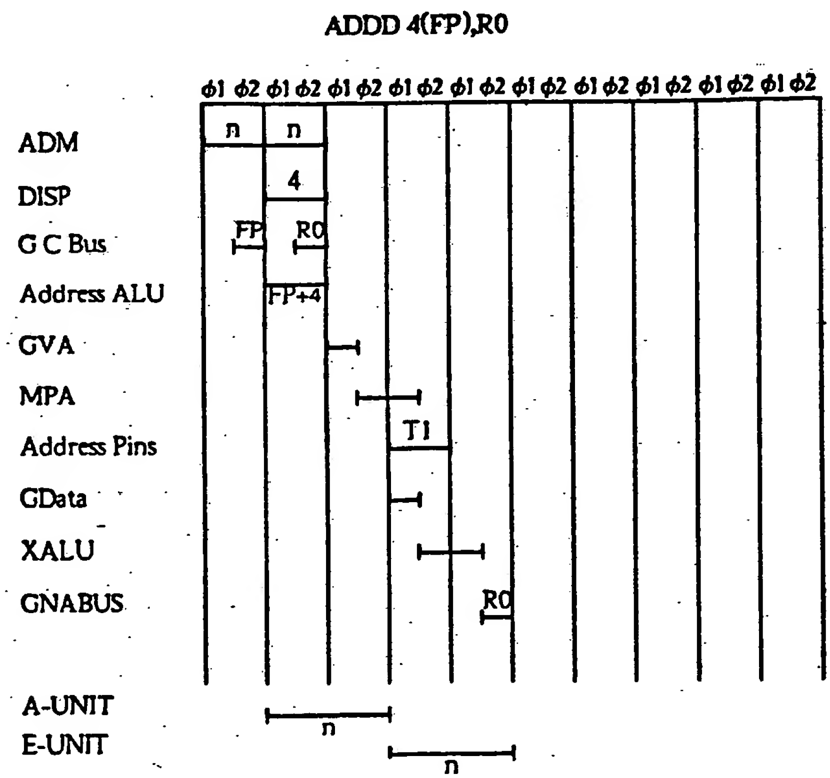


FIG. 8.

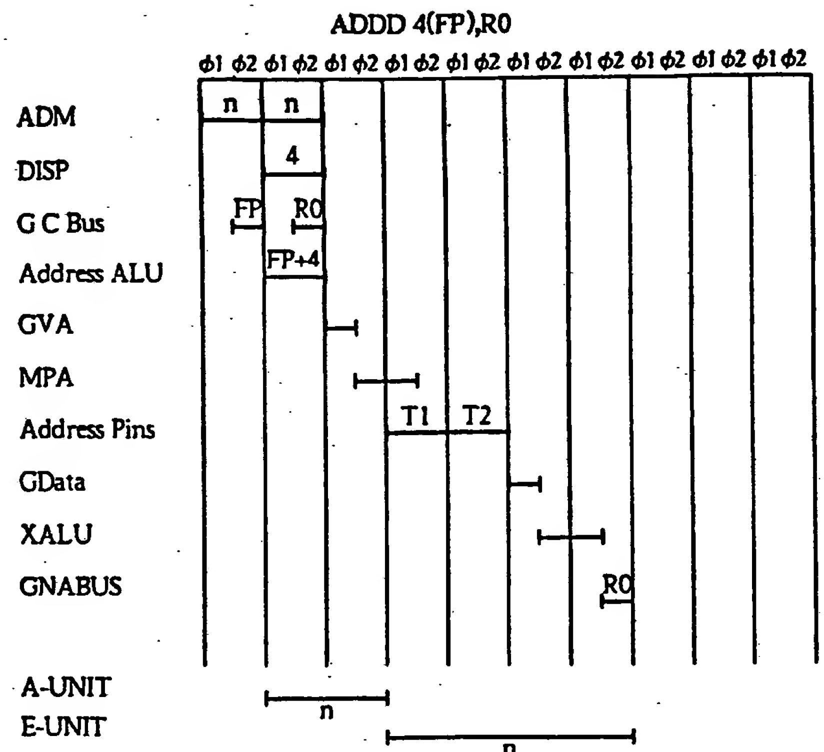
ADDD 4(FP),R0
φ1 φ2 φ1 φ2 φ1 φ2 φ1 φ2 φ1 φ2 φ1 φ2 φ1 φ2 φ1 φ2 φ1 φ2 φ1 φ2
ADM
n
n
DISP
4
G C Bus
FP
R0
Address ALU
FP+4
GVA
MPA
Address Pins
T1
T2
GData
XALU
GNABUS
R0
A-UNIT
n
E-UNIT
n

FIG. 9.

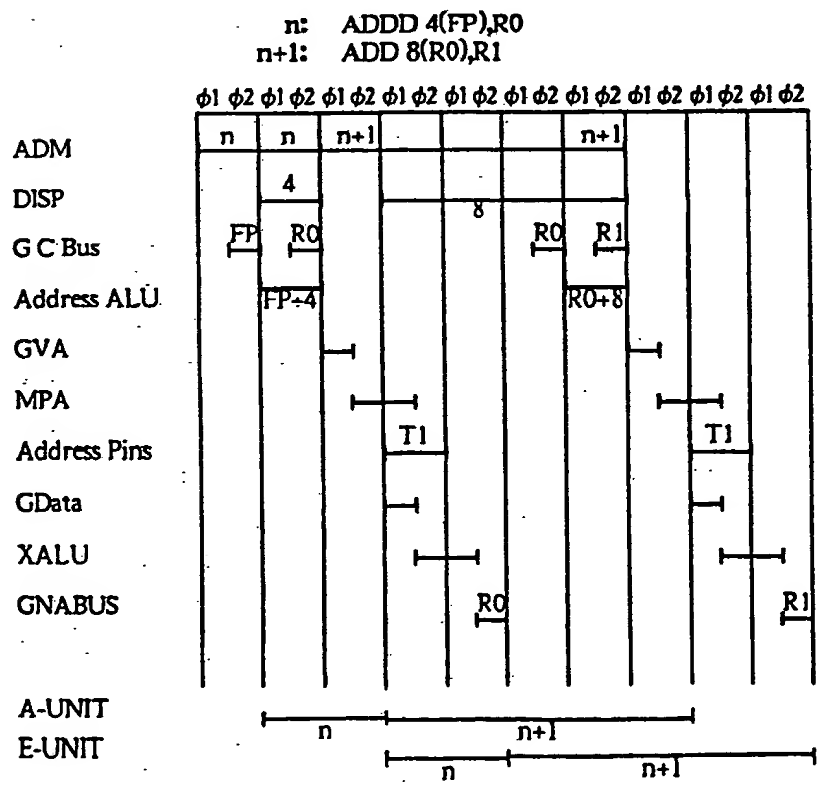
n: ADDD 4(FP),R0
n+1: ADD 8(R0),R1
φ1 φ2 φ1 φ2 φ1 φ2 φ1 φ2 φ1 φ2 φ1 φ2 φ1 φ2 φ1 φ2 φ1 φ2 φ1 φ2
ADM
n
n
n+1
n+1
DISP
4
8
G C Bus
FP
R0
R0
R1
Address ALU
FP+4
R0+8
GVA
MPA
Address Pins
T1
T1
GData
XALU
GNABUS
R0
R1
A-UNIT
n
n+1
E-UNIT
n
n+1

FIG. 10.

2200481

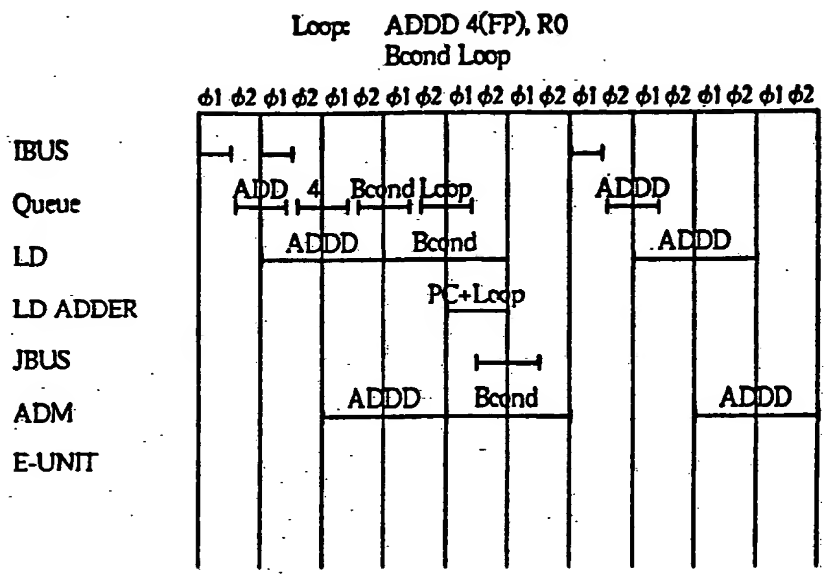


FIG.11.

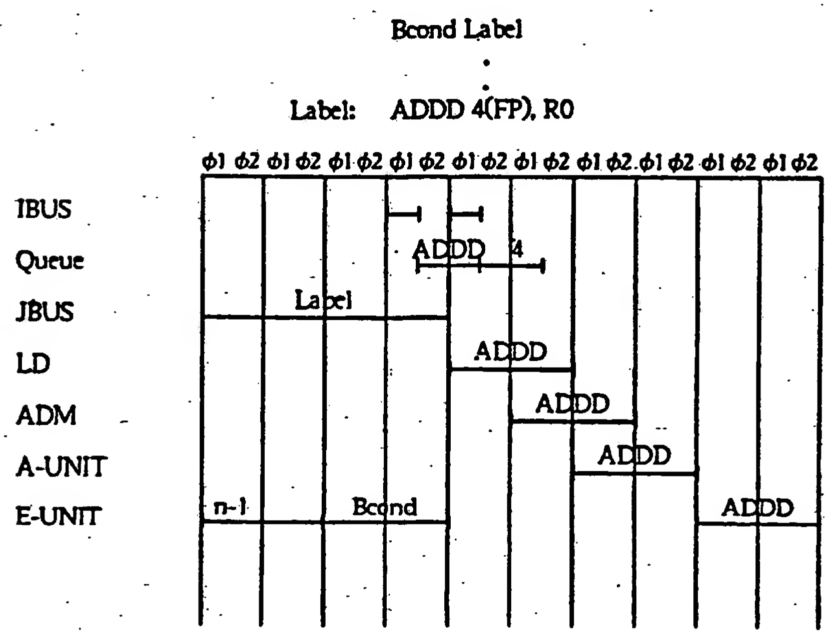


FIG.12

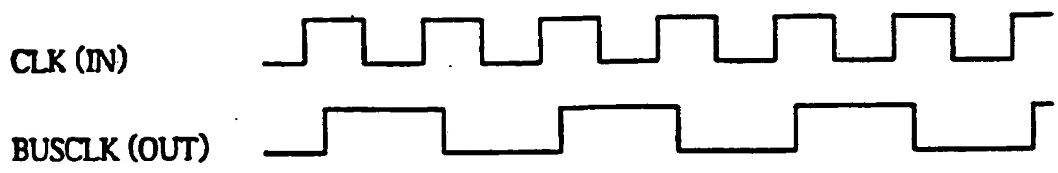
CLK (IN)
BUSCLK (OUT)

FIG.13

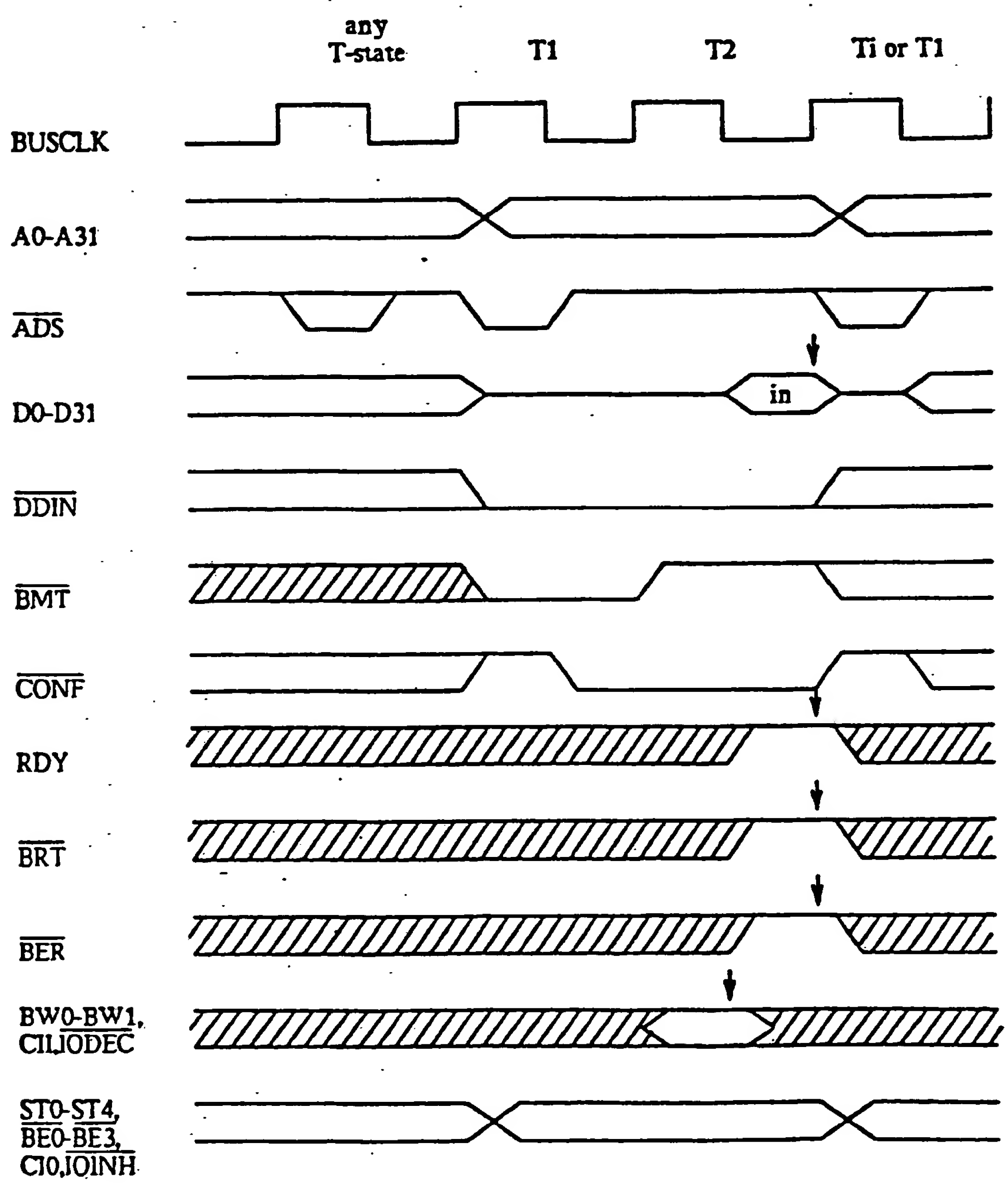
any T-state
T1
T2
Ti or T1
BUSCLK
A0-A31
ADS
D0-D31
in
DDIN
BMT
CONF
RDY
BRT
BER
BW0-BW1, CIL,IODEC
ST0-ST4, BE0-BE3, CI0,IOINH

FIG.14.

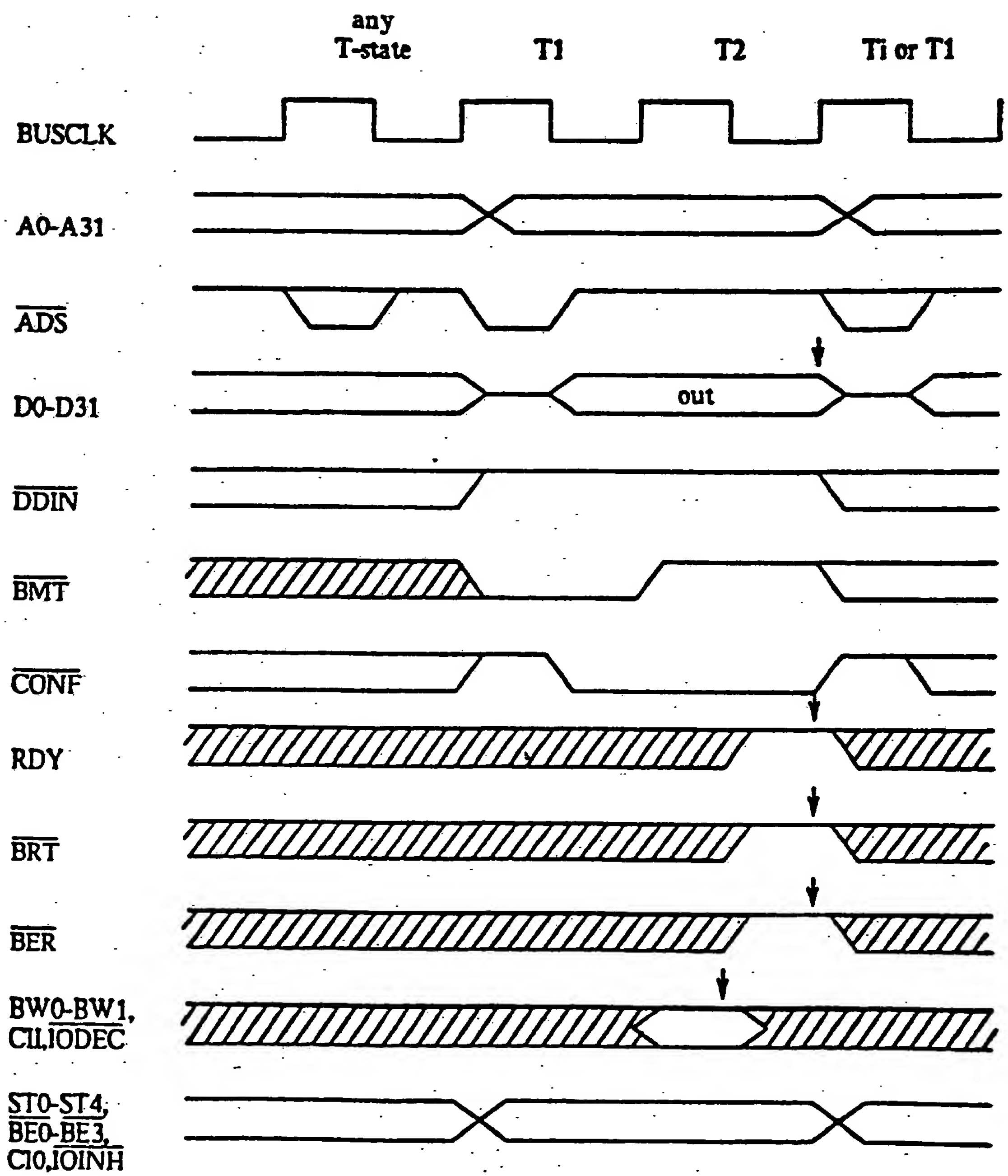
any
T-state
T1
T2
Ti or T1
BUSCLK
A0-A31
ADS
D0-D31
out
DDIN
BMT
CONF
RDY
BRT
BER
BW0-BW1,
CII,IODEC
ST0-ST4,
BE0-BE3,
CI0,IOINH

FIG.15

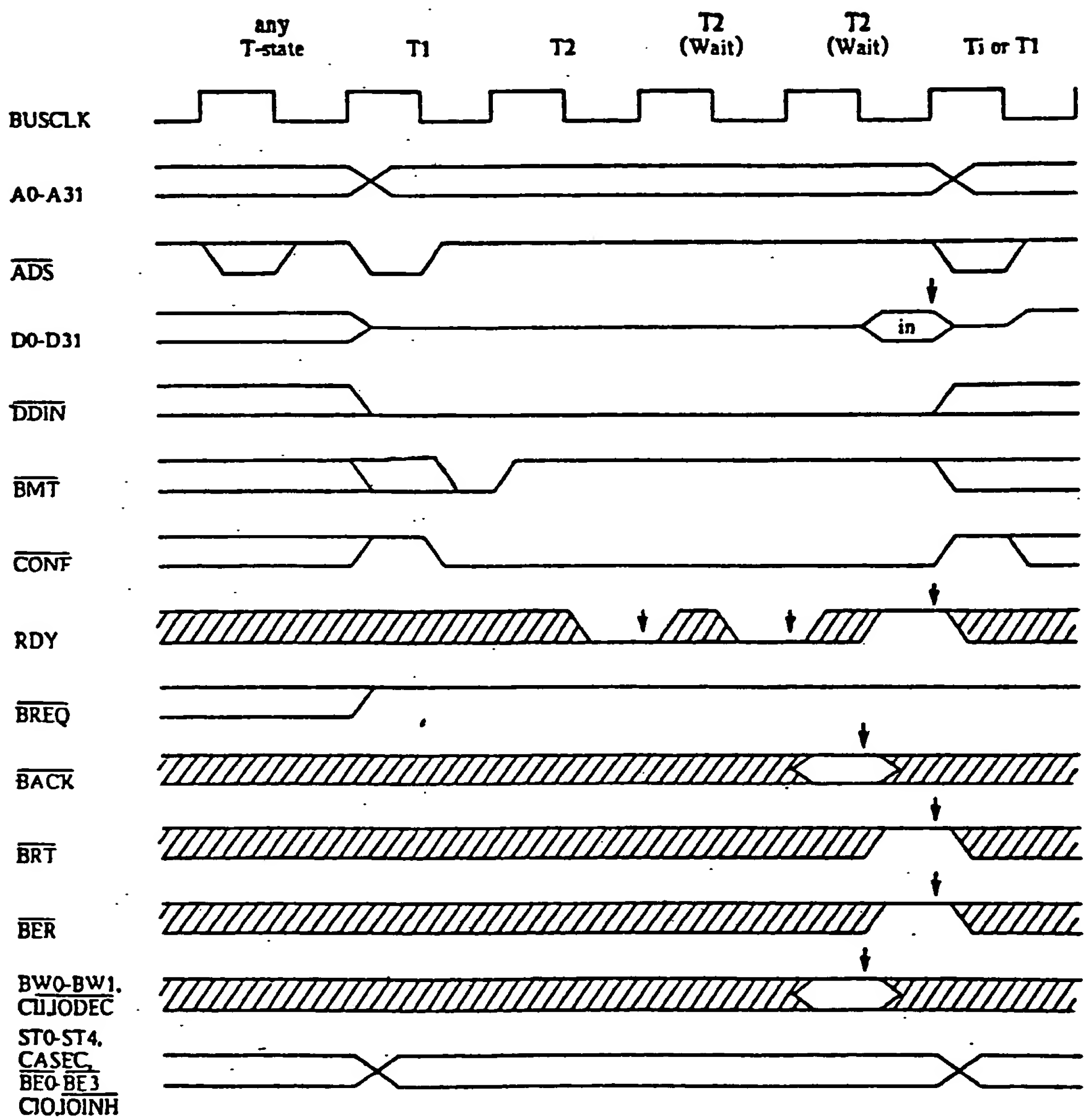
any
T-state
T1
T2
T2
(Wait)
T2
(Wait)
Ti or T1
BUSCLK
A0-A31
ADS
D0-D31
in
DDIN
BMT
CONF
RDY
BREQ
BACK
BRT
BER
BW0-BW1,
CIIJODEC
ST0-ST4,
CASEC,
BE0-BE3
CIOJOINH

FIG.16.

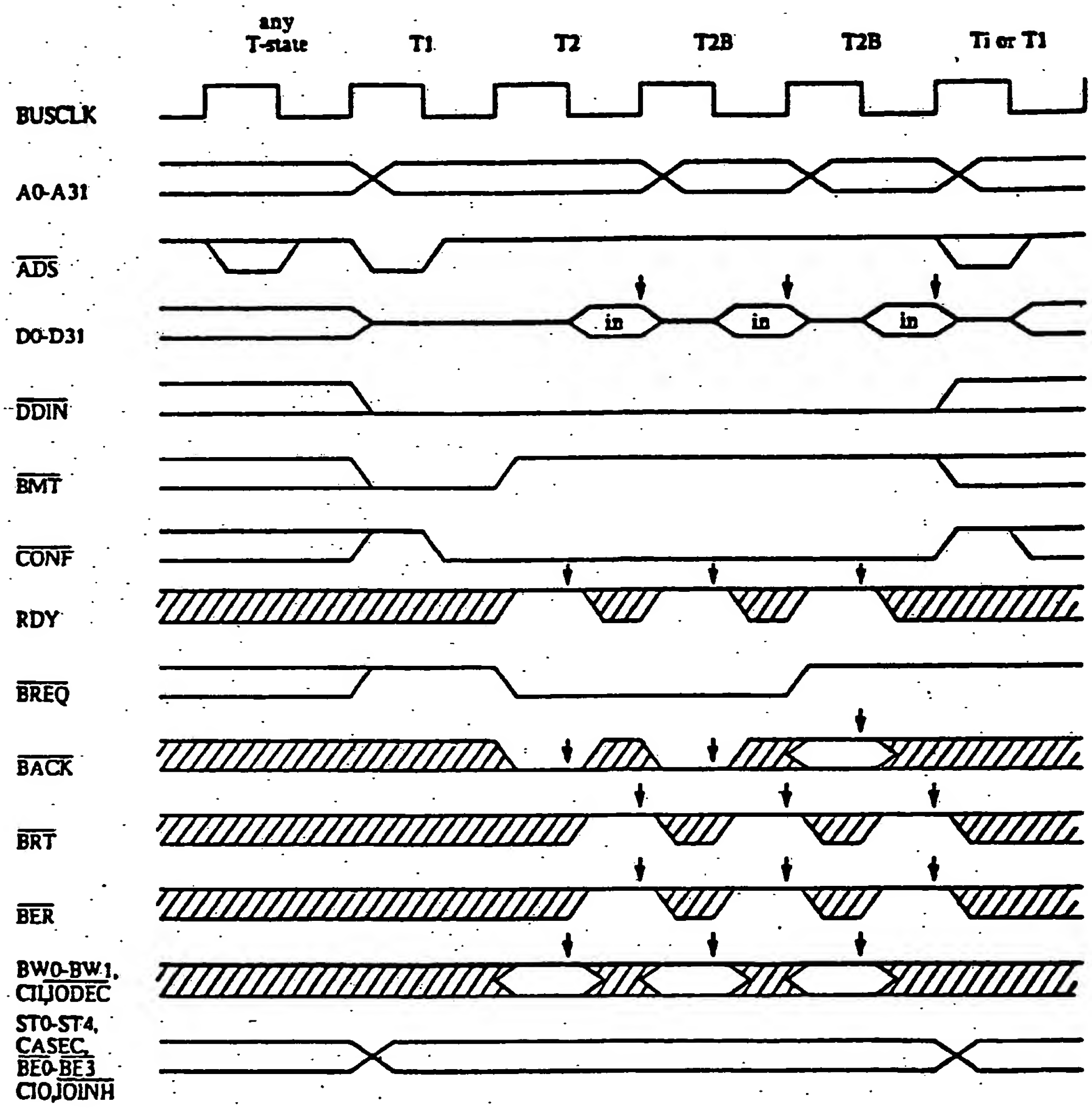
any T-state
T1
T2
T2B
T2B
Ti or T1
BUSCLK
A0-A31
ADS
D0-D31
in
in
in
DDIN
BMT
CONF
RDY
BREQ
BACK
BRT
BER
BW0-BW1, CILJODEC
ST0-ST4, CASEC, BE0-BE3 CIO,JOINH

FIG.17

2200481

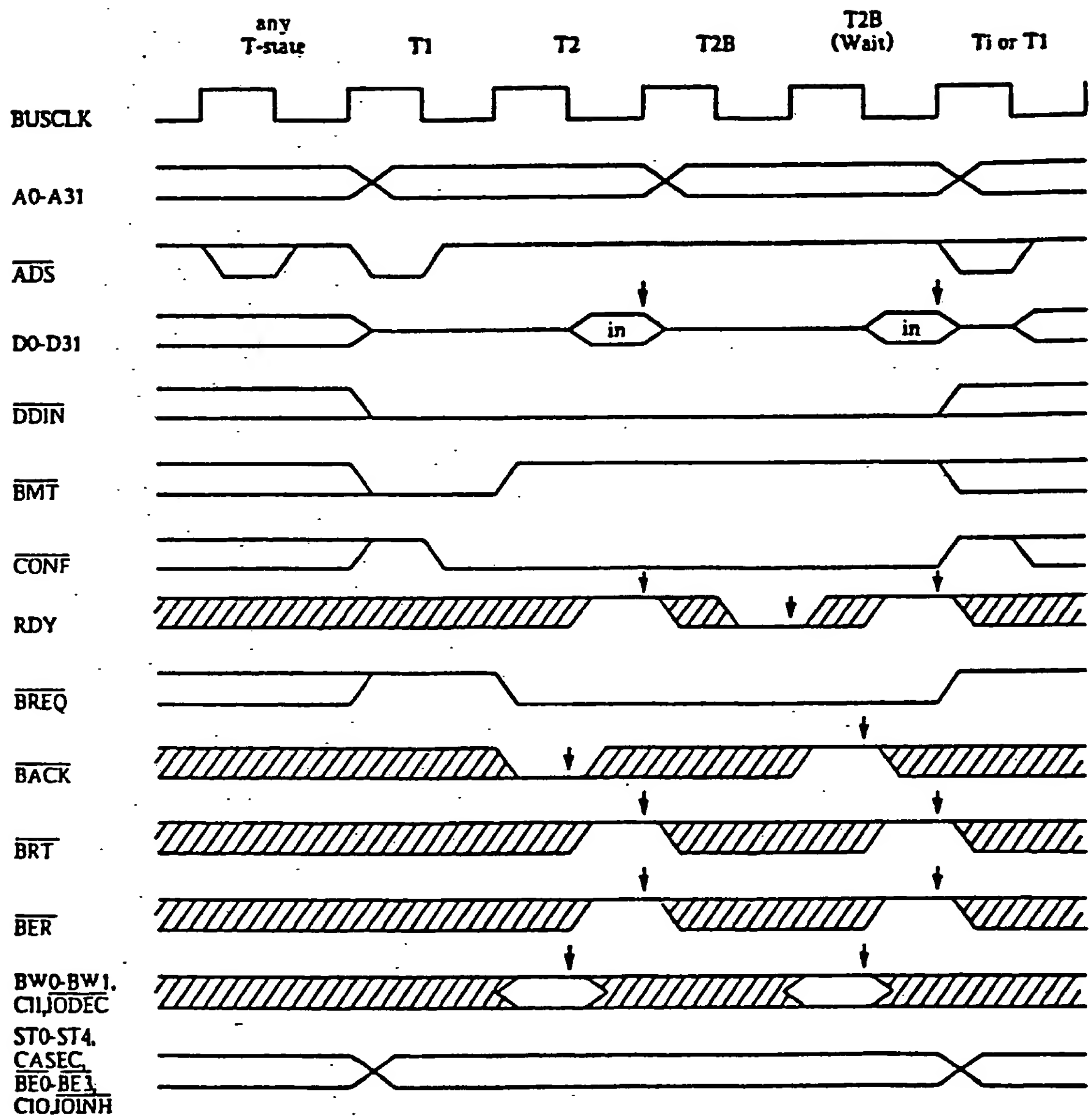


FIG.18

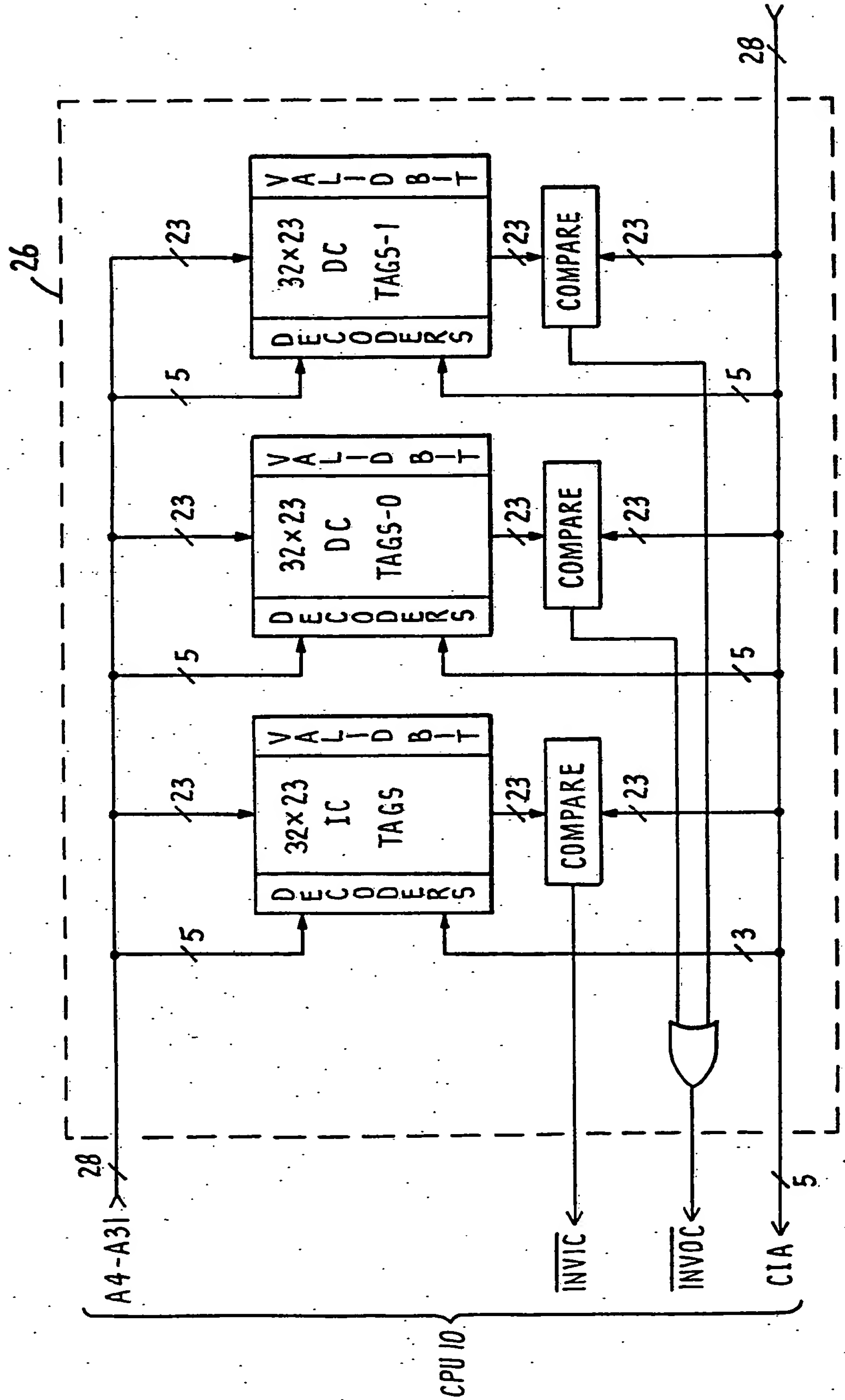
26
28
23
5
3
VALID BIT
32×23
DC
TAGS-1
TAGS-0
IC
TAGS
DECODERS
COMPARE
A4-A31
INVIC
INVOC
CIA
CPU IO

FIG.19.

2200481

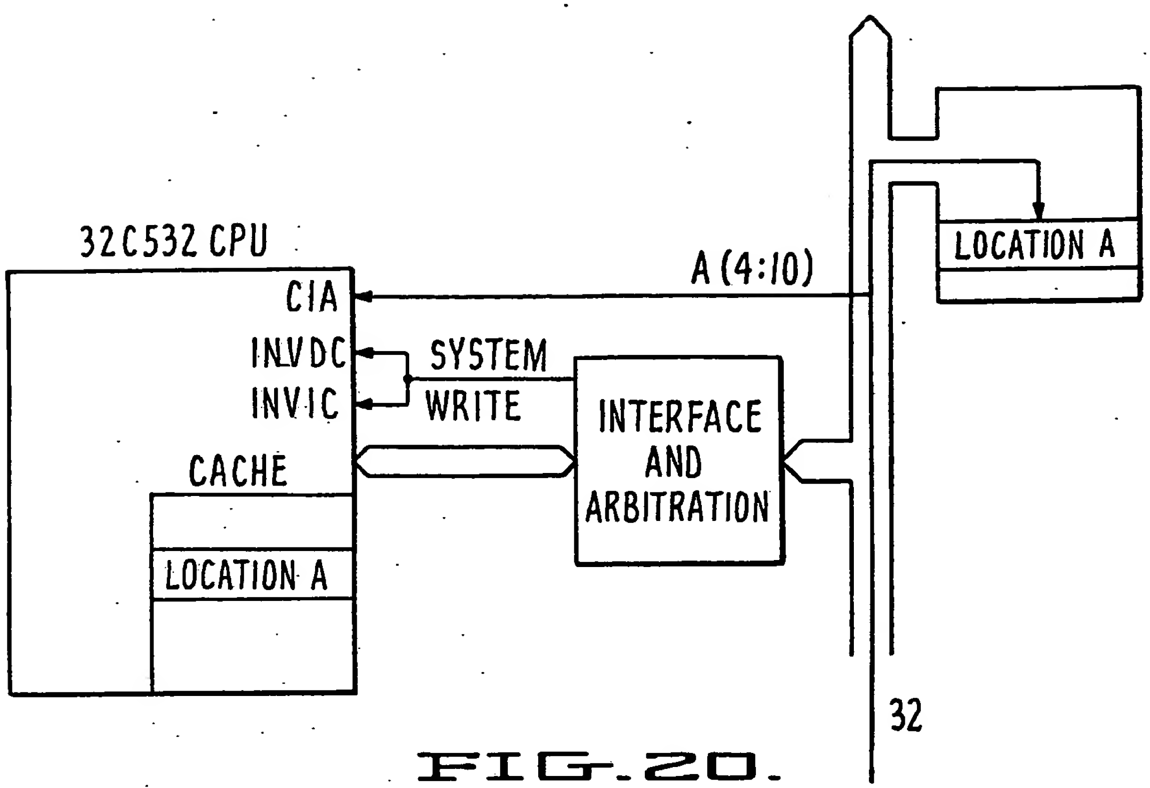


FIG. 20.

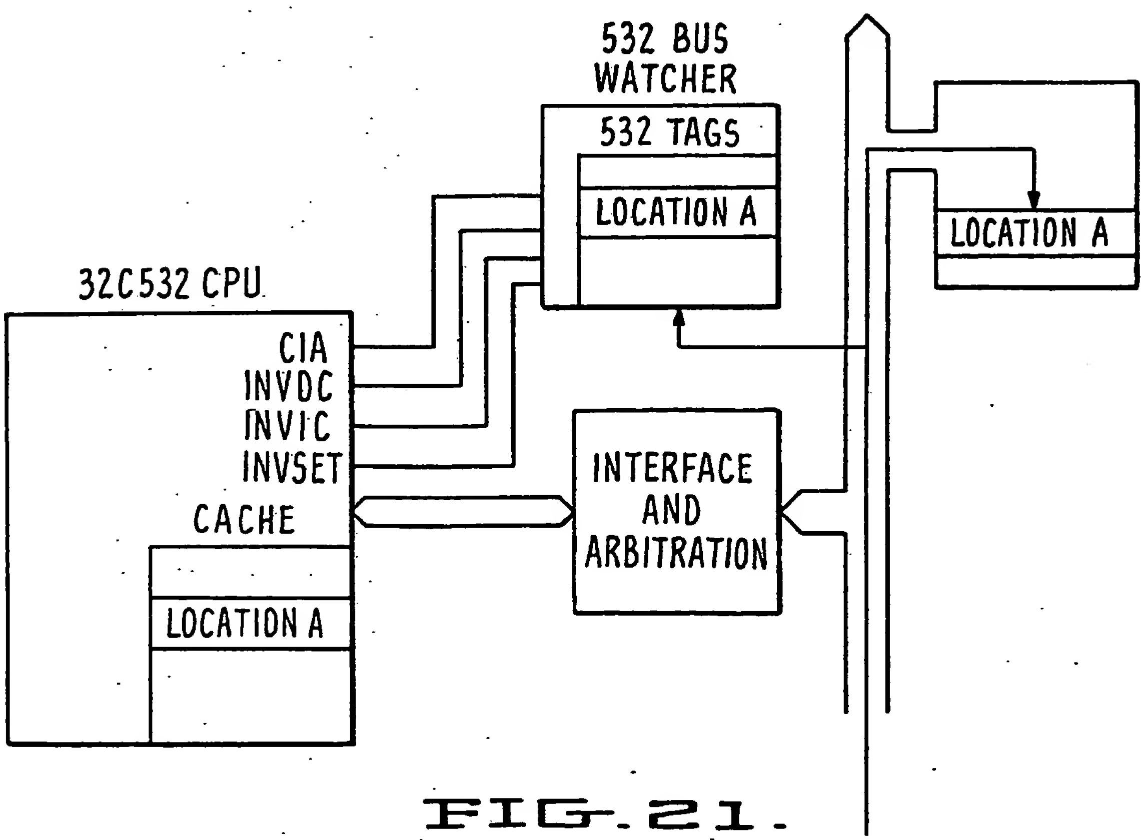


FIG. 21.

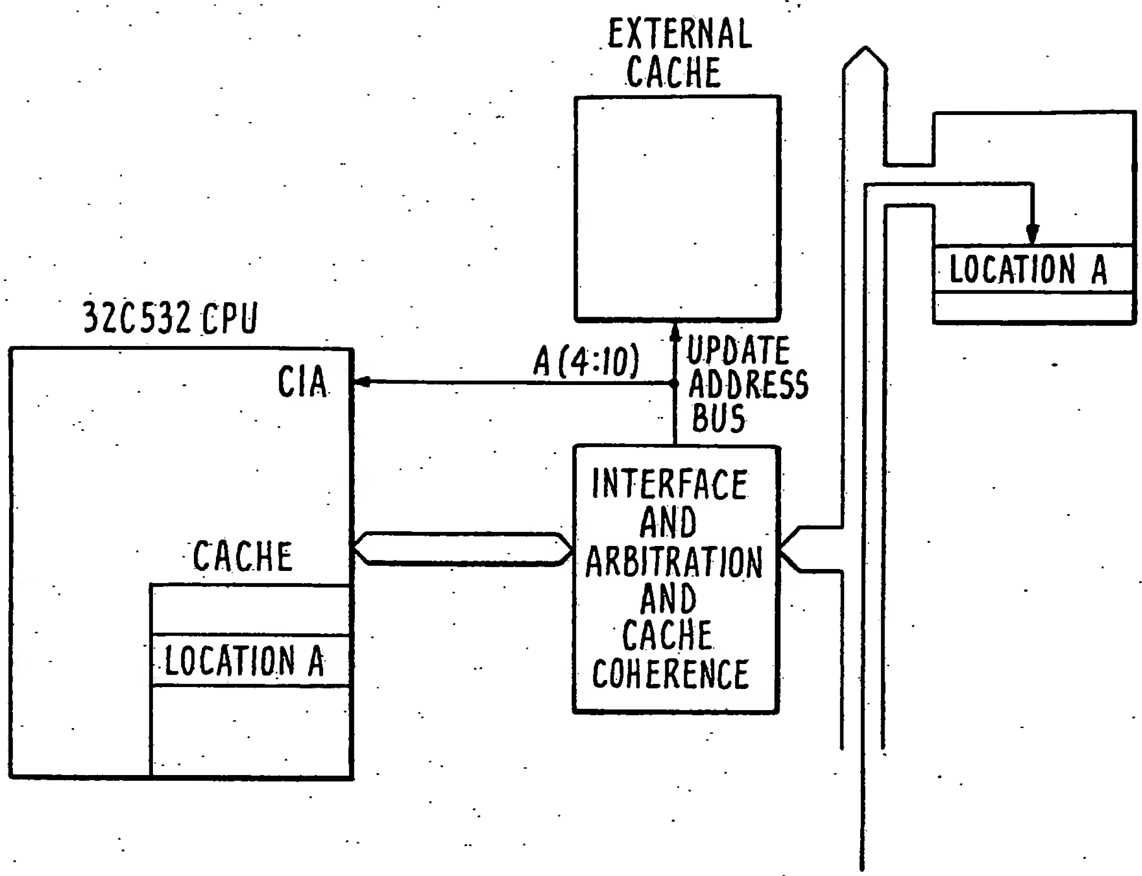
EXTERNAL CACHE
32C532 CPU
CIA
A (4:10)
UPDATE ADDRESS BUS
CACHE
LOCATION A
INTERFACE AND ARBITRATION AND CACHE COHERENCE
LOCATION A

FIG. 22.

MAINTAINING COHERENCE BETWEEN A MICROPROCESSOR'S INTEGRATED CACHE AND EXTERNAL MEMORY

The present invention relates to data processing systems and, in particular, to a method of maintaining coherence between a microprocessor's integrated cache memory and external memory without adversely effecting the microprocessor's performance.

In conventional data processing system architecture, a central processing unit processes instructions and operands which it retrieves from memory via an external interface bus. Because the central processing unit can execute at a rate much faster than the rate at which instructions and operands can be retrieved from external memory, a small high-speed buffer or cache memory is often located between the central processing unit and external memory to minimize the time spent by the central processing unit waiting for instructions and data.

A cache memory dynamically replaces its contents to insure that the most likely to be used information is readily available to the central processing unit. When the central processing unit needs information, it accesses the cache and, if the required information is found within the cache, no access to external memory over the external interface bus need be performed.

Cache memory is a feature which has been introduced only recently to high performance microprocessors. In these microprocessor architectures, however, the cache, while located in the

microprocessor's computing cluster, is not integrated on the same semiconductor "microchip" with the microprocessor. Having a cache memory integrated "on-chip" would provide the advantage of further reducing the time delay inherent in going "off-chip" for information. Integrated cache is essential for achieving top performance from a microprocessor.

To achieve top performance and correct operation, a cache memory must reflect the most up-to-date information which may be needed by the central processing unit. This requirement is usually termed "maintaining cache coherence". Maintaining cache coherence can be summarized by the following sequence of events. First, the cache receives a copy of an information character from an address within external memory. The information character at that address in external memory is then modified by a write from an external device. As a result, a "stale" character exists in the cache. To maintain coherence between external memory and the cache, the stale character in the cache must either be updated or invalidated before the central processing unit requests information from the corresponding address.

In conventional microprocessor designs which use off-chip caches, cache entry invalidations are performed by presenting the addresses for modified locations in external memory to a set of cache address tags for comparison. In some cases, an extra set of cache tags is used to avoid interference with the microprocessor's cache references. This comparison and the resultant cache invalidation, if any, are performed via the system interface bus.

However, if conventional cache entry invalidation techniques are applied to an integrated cache, a number

of problems arise. First, additional pins may be required for the microprocessor to input invalidation addresses. If additional pins are not added and instead the pins for the microprocessor's external references are also used for input of cache invalidation addresses, then contention is created for the interface bus and microprocessor performance is degraded. Second, an additional copy of the address tags for the on-chip cache may be required for comparison with the invalidation addresses. Otherwise, if only one set of tags is used, there would be contention for the tags and, again, performance would suffer.

Accordingly, it is an object of the present invention to provide a method for maintaining coherence in an integrated cache memory without adversely effecting the performance of the associated central processing unit.

It is also an object of the present invention to provide a method for monitoring externally the contents of an on-chip cache.

It is a further object of the present invention to provide for selective invalidation of on-chip cache locations.

The solution provided by the present invention to the above-described problems is to limit the number of pins on the microprocessor's interface by specifying the location in the cache to invalidate; that is, the cache set to be invalidated is specified rather than the main memory address. By using a separate invalidation bus, cache invalidations can occur without interfering with the microprocessor's external

references. By using dual-ported validity bits in the on-chip cache, invalidations can occur without interfering with the microprocessor's on-chip references.

"Bus-Watcher" circuitry is provided which contains the additional copy of the cache tags, rather than placing the tags on the microprocessor. This reduces the cost of the microprocessor. Also, the Bus-Watcher is not required when the rate of invalidations is low. Whenever a location in main memory is modified, it is possible to invalidate the cache set containing that location even if another address is stored in the cache. This saves the cost of the special Bus-Watcher, but reduces performance because unnecessary invalidations are performed. The cost/performance tradeoffs regarding whether to include a Bus-Watcher are left to the system designer.

The on-chip cache of the microprocessor described herein includes a 512-byte Instruction Cache and a separate 1024-byte Data Cache. The Instruction Cache and the Data Cache may be separately enabled. The contents of the two caches can be optionally locked to fixed memory locations. By providing the option of locking specific locations into the caches, the central processing unit offers very fast on-chip access to critical instructions and data, which can be of great benefit in real-time applications.

A cache invalidation instruction can be executed to either entirely invalidate the Instruction Cache and/or Data Cache or an invalidation instruction can be executed to invalidate only a single 16-byte block in either or both caches.

The use of the caches can be inhibited for individual locations using a cache inhibit input signal

which indicates to the central processing unit that the memory reference of the current bus cycle is not cacheable.

Figure 1 is a schematic block diagram illustrating a general microprocessor architecture which utilizes a method for maintaining cache coherence in accordance with the present invention.

Figure 2 is a schematic diagram illustrating the interface signals of the microprocessor described herein.

Figure 3 is a schematic block diagram illustrating the major functional units and interconnecting buses of the microprocessor described herein.

Figure 4 is a schematic block diagram illustrating the structure of the integrated Instruction Cache of the microprocessor described herein.

Figure 5 is a schematic block diagram illustrating the structure of the integrated Data Cache of the microprocessor described herein.

Figure 6 is a timing diagram illustrating the timing sequence for access to the Data Cache.

Figure 7 is a schematic diagram illustrating the general structure of the 4-stage Pipeline of the microprocessor described herein.

Figure 8 is a timing diagram illustrating Pipeline timing for an internal Data Cache hit.

Figure 9 is a timing diagram illustrating Pipeline timing for an internal Data Cache miss.

Figure 10 is a timing diagram illustrating the effect of an address-register interlock on Pipeline timing.

Figure 11 is a timing diagram illustrating the effect of correctly predicting a branch instruction to be taken in the operation of the microprocessor described herein.

Figure 12 is a timing diagram illustrating the effect of incorrectly predicting the resolution of a branch instruction in the operation of the microprocessor described herein.

Figure 13 is a timing diagram illustrating the relationship between the CLK input and BUSCLK output signals of the microprocessor described herein.

Figure 14 is a timing diagram illustrating the basic read cycle of the microprocessor described herein.

Figure 15 is a timing diagram illustrating the basic write cycle of the microprocessor described herein.

Figure 16 is a timing diagram illustrating a read cycle of the microprocessor described herein extended with two wait cycles.

Figure 17 is a timing diagram illustrating a burst read cycle, having three transfers, which is terminated by the microprocessor described herein.

Figure 18 is a timing diagram illustrating a burst read cycle terminated by the microprocessor described herein, the burst cycle having two transfers, the second transfer being extended by one wait state.

Figure 19 is a schematic block diagram illustrating a Bus Watcher used to maintain cache coherence in accordance with the present invention.

Figure 20 is a schematic block diagram illustrating a cache coherence solution for a low invalidation rate system.

Figure 21 is a schematic block diagram illustrating a cache coherence solution for a high invalidation rate system.

Figure 22 is a schematic block diagram illustrating a cache coherence solution for a high invalidation rate system with a large external cache memory.

Fig. 1 shows the general architecture of a microprocessor (CPU) 10 which implements a method for maintaining coherence in an integrated cache memory in accordance with the present invention.

CPU 10 initiates bus cycles to communicate with external memory and other devices in the system to fetch instructions, read and write data, perform floating-point operations and respond to exception requests.

CPU 10 includes a 4-stage instruction Pipeline 12 that is capable of executing, at 20 MHz, up to 10 MIPS (millions of instructions per second). Also, integrated on-chip with the instruction Pipeline 12 are three storage buffers that sustain the heavy demand of Pipeline 12 for instructions and data. The storage buffers include a 512-byte Instruction Cache 14, a 1024-byte Data Cache 16 and a 64-entry translation buffer which is located within an integrated memory management unit (MMU) 18. The primary functions of MMU 18 are to arbitrate requests for memory references and to translate virtual addresses to physical addresses. An integrated Bus Interface Unit (BIU) 20 controls the bus cycles for external references.

Placing the cache and memory management functions on the same chip with instruction Pipeline 12 provides

excellent cost/performance by improving memory access time and bandwidth for all applications.

Both Instruction Cache 14 and Data Cache 16 are physical. This is important in order to support cache coherence with external caches and memories. In multiprocessor systems, or in direct memory access (DMA) operations in all systems, data may be written to an external memory while the same address exists in the internal caches 14,16 and needs, therefore, to be invalidated. If the internal caches 14,16 were virtual, a single cache entry would be very difficult to invalidate since the external address is physical. Physical caches allow for single entry invalidation.

CPU 10 is also compatible with available peripheral devices, such as Interrupt Control Unit (ICU) 24 (e.g., NS32202). The ICU interface to CPU 10 is completely asynchronous, so it is possible to operate ICU 24 at lower frequencies than CPU 10.

CPU 10 incorporates its own clock generator. Therefore, no timing control unit is required.

CPU 10 also supports both external cache memory 25 as well as "Bus-Watcher" circuitry 26, described in detail below, which assists in maintaining internal cache coherence. As shown in Fig. 2, CPU 10 has 114 interface signals for bus timing and control, cache control, exception requests and other functions. The following list provides a summary of the CPU 10 interface signal functions:

<u>Input Signals</u>

| | |
|---|---|
| $\overline{\text{BACK}}$ | Burst Acknowledge (Active Low). When active in response to a burst request, indicates that the memory supports burst cycles. |

$\overline{\text{BER}}$ Bus Error (Active Low). Indicates to CPU 10 that an error was detected during the current bus cycle.

$\overline{\text{BRT}}$ Bus Retry (Active Low). Indicates that CPU 10 must perform the current bus cycle again.

BWO-BW1 Bus Width (2 encoded lines). These lines define the bus width (8, 16 or 32 bits) for each data transfer, as shown in Table 1.

| BW1 | BWO | Bus Width |
|---|---|---|
| 0 | 0 | reserved |
| 0 | 1 | 8 bits |
| 1 | 0 | 16 bits |
| 1 | 1 | 32 bits |

Table 1

CIA0-CIA6 Cache Invalidation Address (7 encoded lines) The cache invalidation address is presented on the CIA bus. Table 2 presents the CIA lines relevant for each of the internal caches of CPU 10.

| | |
|---|---|
| CIA (0:4) | Set address in DC and IC |
| CIA (5:6) | Reserved |

Table 2

CII Cache Inhibit In (Active High). Indicates to CPU 10 that the memory reference of the current bus cycle is not cacheable.

$\overline{\text{CINVE}}$ Cache Invalidation Enable. Input which determines whether the External Cache Invalidation options or the Test Mode operation have been selected.

CLK Clock. Input clock used to derive all timing for CPU 10.

| | |
|---|---|
| $\overline{\text{DBG}}$ | Debug Trap Request (Falling-Edge Activated). High-to-low transition of this signal causes Trap (DBG). |
| $\overline{\text{HOLD}}$ | Hold Request (Active Low). Requests CPU 10 to release the bus for DMA or multiprocessor purposes. |
| $\overline{\text{INT}}$ | Interrupt (Active Low). Maskable interrupt request. |
| $\overline{\text{INVSET}}$ | Invalidate Set (Active Low). When Low, only a set in the on-chip caches is invalidated; when High, the entire cache is invalidated. |
| $\overline{\text{INVDC}}$ | Invalidate Data Cache (Active Low). When low, an invalidation is done in the Data Cache. |
| $\overline{\text{INVIC}}$ | Invalidate Instruction Cache (Active Low). When low, an invalidation is done in the Instruction Cache. |
| $\overline{\text{IODEC}}$ | I/O Decode (Active Low). Indicates to CPU 10 that a peripheral device is addressed by the current bus cycle. |
| $\overline{\text{NMI}}$ | Nonmaskable Interrupt (Falling-Edge Activated). A High-to-Low transition of this signal requests a nonmaskable interrupt. |
| RDY | Ready (Active High). While this signal is not active, CPU 10 extends the current bus cycle to support a slow memory or peripheral device. |
| $\overline{\text{RST}}$ | Reset (Active Low). Generates reset exceptions to initialize CPU 10. |
| $\overline{\text{SDONE}}$ | Slave Done (Active Low). Indicates to CPU 10 that a Slave Processor has completed executing an instruction. |

| | |
|---|---|
| $\overline{\text{STRAP}}$ | Slave Trap (Active Low). Indicates to CPU 10 that a Slave Processor has detected a trap condition while executing an instruction. |
| <u>Output Signals</u> | |
| A0-A31 | Address Bus (3-state, 32 lines) Transfers the 32-bit address during a bus cycle; A0 transfers the least significant bit. |
| $\overline{\text{ADS}}$ | Address Strobe (Active Low, 3-State). Indicates that a bus cycle has begun and a valid address is on the address bus. |
| $\overline{\text{BE0}}$-$\overline{\text{BE3}}$ | Byte Enables (Active Low, 3-state, 4 lines). Signals enabling transfer on each byte of the data bus, as shown in Table 3. |

| BE | Enables Bits |
|---|---|
| 0 | 0 - 7 |
| 1 | 8 - 15 |
| 2 | 16 - 23 |
| 3 | 24 - 31 |

Table 3

| | |
|---|---|
| $\overline{\text{BMT}}$ | Begin Memory Transaction (Active Low, 3-State). Indicates that the current bus cycle is valid, that is, the bus cycle has not been cancelled; Available earlier in the bus cycle than CONF. |
| $\overline{\text{BP}}$ | Break Point (Active Low). Indicates that CPU 10 has detected a debug condition. |
| $\overline{\text{BREQ}}$ | Burst Request (Active Low, 3-state). Indicates that CPU 10 is requesting to perform burst cycles. |
| BUSCLK | Bus Clock Output clock for bus timing. |

| | |
|---|---|
| CASEC | Cache Section (3-state) For cacheable data read bus cycles, indicates the section of the on-chip Data Cache 18 into which the data will be placed. |
| CIO | Cache Inhibit (Active High). Indication by CPU 10 that the memory reference of the current bus cycle is not cacheable; Controlled by the CI-bit in the level-2 Page Table Entry. |
| $\overline{\text{CONF}}$ | Confirm Bus Cycle (Active Low, 3-state). Indicates that a bus cycle initiated with ADS is valid; that is, the bus cycle has not been cancelled. |
| $\overline{\text{DDIN}}$ | Data Direction In (Active Low, 3-state). Indicates the direction of transfers on the data bus; when Low during a bus cycle, indicates that CPU 10 is reading data; when High during a bus cycle, indicates that CPU 10 is writing data. |
| $\overline{\text{HLDA}}$ | Hold Acknowledge (Active Low). Activated by CPU 10 in response to the 1-HOLD input to indicate that CPU 10 has released the bus. |
| $\overline{\text{ILO}}$ | Interlocked Bus Cycle (Active Low). Indicates that a sequence of bus cycles with interlock protection is in progress. |
| $\overline{\text{IOINH}}$ | I/O Inhibit (Active Low). Indicates that the current bus cycle should be ignored if a peripheral device is addressed. |
| $\overline{\text{ISF}}$ | Internal Sequential Fetch. Indicates, along with PFS, that the instruction beginning execution is sequential (ISF = Low) or non-sequential (ISF = High). |
| $\overline{\text{PFS}}$ | Program Flow Status (Active Low). A pulse on this signal indicates the beginning of execution for each instruction. |

| | |
|---|---|
| $\overline{SPC}$ | Slave Processor Control (Active Low). Data Strobe for Slave Processor bus cycles. |
| ST0-ST4 | Status (5 encoded lines). Bus cycle status code; ST0 is the least significant bit. The encoding is shown in Table 4. |
| U/$\overline{S}$ | User/Supervisor (3_state). Indicates User_(U/S = High) or Supervisor (U/S = Low) Mode. |

Bidirectional Signals

| | |
|---|---|
| D0-D31 | Data Bus (3-state,32 lines). Transfers 8, 16, or 32 bits of data during a bus cycle; D0 transfers the least significant bit. |

| STATUS 4 | 3 | 2 | 1 | 0 | DESCRIPTION |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | Idle |
| 0 | 0 | 0 | 0 | 1 | Idle: Wait Instruction |
| 0 | 0 | 0 | 1 | 0 | Idle: Halted |
| 0 | 0 | 0 | 1 | 1 | Idle: Waiting for Slave |
| 0 | 0 | 1 | 0 | 0 | Interrupt acknowledge, Master |
| 0 | 0 | 1 | 0 | 1 | Interrupt acknowledge, Cascaded |
| 0 | 0 | 1 | 1 | 0 | End of Interrupt, Master |
| 0 | 0 | 1 | 1 | 1 | End of Interrupt, Cascaded |
| 0 | 1 | 0 | 0 | 0 | Sequential Instruction Fetch |
| 0 | 1 | 0 | 0 | 1 | Non-sequential Instruction Fetch |
| 0 | 1 | 0 | 1 | 0 | Data transfer |
| 0 | 1 | 0 | 1 | 1 | Read Read-Modify-Write Operand |
| 0 | 1 | 1 | 0 | 0 | Read for Effective address |
| 0 | 1 | 1 | 0 | 1 | Access PTE1 by MMU |
| 0 | 1 | 1 | 1 | 0 | Access PTE2 by MMU |
| 0 | 1 | 1 | 1 | 1 | reserved |
| 1 | 0 | 0 | 0 | 0 | reserved |
| 1 | 0 | 0 | 0 | 1 | reserved |
| 1 | 0 | 0 | 1 | 0 | reserved |
| 1 | 0 | 0 | 1 | 1 | reserved |
| 1 | 0 | 1 | 0 | 0 | reserved |
| 1 | 0 | 1 | 0 | 1 | reserved |
| 1 | 0 | 1 | 1 | 0 | reserved |
| 1 | 0 | 1 | 1 | 1 | reserved |
| 1 | 1 | 0 | 0 | 0 | reserved |
| 1 | 1 | 0 | 0 | 1 | reserved |
| 1 | 1 | 0 | 1 | 0 | reserved |
| 1 | 1 | 0 | 1 | 1 | reserved |
| 1 | 1 | 1 | 0 | 0 | reserved |
| 1 | 1 | 1 | 0 | 1 | Transfer Slave Processor Operand |
| 1 | 1 | 1 | 1 | 0 | Read Slave Processor Status |
| 1 | 1 | 1 | 1 | 1 | Broadcast Slave ID + Opcode |

Table 4

Referring to Fig. 3, CPU 10 is organized internally as eight major functional units that operate in parallel to perform the following operations to execute instructions: prefetch, decode, calculate effective addresses and read source operands, calculate

results and store to registers, store results to memory.

A Loader 28 prefetches instructions and decodes them for use by an Address Unit 30 and an Execution Unit 32. Loader 28 transfers instructions received from Instruction Cache 14 on the IBUS bus into an 8-byte instruction queue. Loader 28 can extract an instruction field on each cycle, where a "field" means either an opcode (1 to 3 bytes including addressing mode specifiers), displacement or immediate value. Loader 28 decodes the opcode to generate the initial microcode address, which is passed on the LADR bus to Execution Unit 32. The decoded general addressing modes are passed on the ADMS bus to Address Unit 30. Displacement values are passed to Address Unit 30 on the DISP bus. Immediate values are available on the GCBUS bus. Loader 28 also includes a branch-prediction mechanism, which is described in greater detail below.

Address Unit 30 calculates effective addresses using a dedicated 32-bit adder and reads source operands for Execution Unit 32. Address Unit 30 controls a port from a Register File 34 to the GCBUS through which it transfers base and index values to the address adder and data values to Execution Unit 32. Effective addresses for operand references are transferred to MMU 18 and Data Cache 16 on the GVA bus, which is the virtual address bus.

Execution Unit 32 includes the data path and the microcoded control for executing instructions and processing exceptions. The data path includes a 32-bit Arithmetic Logic Unit (ALU), a 32-bit barrel shifter, an 8-bit priority encoder, and a number of counters. Special-purpose hardware incorporated in Execution Unit 32 supports multiplication, retiring one bit per cycle

with optimization for multipliers of small absolute value.

Execution Unit 32 controls a port to Register File 34 from the GNA bus on which it stores results. The GNA bus is also used by Execution Unit 32 to read values of dedicated registers, like configuration and interrupt base registers, which are included in Register File 34. A 2-entry data buffer allows Execution Unit 32 to overlap the execution of one instruction with storing results to memory for previous instructions. The GVA bus is used by Execution Unit 32 to perform memory references for complex instructions (e.g., string operations) and exception processing.

Register File 34 is dual-ported, allowing read access by Address Unit 30 on the GCBUS and read/write access by Execution Unit 32 on the GNA bus. Register File 34 holds the general-purpose registers, dedicated registers, and program counter values for Address Unit 30 and Execution Unit 32.

MMU 18 is compatible with the memory management functions of CPU 10. Instruction Cache 14, Address Unit 30 and Execution Unit 32 make requests to MMU 18 for memory references. MMU 18 arbitrates the requests, granting access to transfer a virtual address on the GVA bus. MMU 18 translates the virtual address it receives on the GVA bus to the corresponding physical address, using the translation buffer. MMU 18 transfers the physical address on the MPA bus to either Instruction Cache 14 or Data Cache 16, depending on whether an instruction or data reference is being performed. The physical address is also transferred to BIU 20 for an external bus cycle.

Bus Interface Unit (BIU) 20 controls the bus cycles for references by Instruction Cache 14, Address

Unit 30 and Execution Unit 32. BIU 20 contains a 3-entry buffer for external references. Thus, for example, BIU 20 can be performing a bus cycle for an instruction fetch while holding the information for another bus cycle to write to memory and simultaneously accepting the next data read.

Referring to Fig. 4, Instruction Cache 14 stores 512 bytes in a direct-map organization. Bits 4 through 8 of a reference instruction's address select 1 of 32 sets. Each set contains 16 bytes of code and a log that holds address tags comprising the 23 most-significant bits of the physical address for the locations stored in that set. A valid bit is associated with every double-word.

Instruction Cache 14 also includes a 16-byte instruction buffer from which it can transfer 32-bits of code per cycle on the IBUS to Loader 28. In the event that the desired instruction is found in Instruction Cache 14 (a "hit"), the instruction buffer is loaded directly from the selected set of Instruction Cache 14 and no bus cycle is required with external memory. In the event that a referenced instruction is not found in Instruction Cache 14 (a "miss"), Instruction Cache 14 transfers the address of the missing double-word on the GVA bus to MMU 18, which translates the address for BIU 20. BIU 20 initiates a burst read cycle to load the instruction buffer from external memory through the GBDI bus. The instruction buffer is then written to one of the sets of Instruction Cache 14.

Instruction Cache 14 holds counters for both the virtual and physical addresses from which to prefetch the next double-word of the instruction stream. When Instruction Cache 14 must begin prefetching from a new

instruction stream, the virtual address for the new stream is transferred from Loader 28 on the JBUS. When crossing to a new page, Instruction Cache 14 transfers the virtual address to MMU 18 on the GVA bus and receives back the physical address on the MPA bus.

Instruction Cache 14 supports an operating mode to lock its contents to fixed locations. This feature is enabled by setting a Lock Instruction Cache (LIC) bit in the configuration register. It can be used in real-time systems to allow fast, on-chip access to the most critical routines. Instruction Cache 14 can be enabled by setting an Instruction Cache Enable (IC) bit in the configuration register.

Data Cache 16 stores 1024 bytes of data in a two-way set associative organization, as shown in Fig. 5. Each set has two entries containing 16 bytes and two address tags that hold the 23 most significant bits of the physical address for the locations stored in the two entries. A valid bit is associated with every double-word.

The timing to access Data Cache 16 is shown in Fig. 6. First, virtual address bits 4 through 8 on the GVA bus are used to select the appropriate set within Data Cache 16 to read the two entries. Simultaneously, MMU 18 is translating the virtual address and transferring the physical address to Data Cache 16 and BIU 20 on the MPA bus. Data Cache 16 compares the two address tags with the physical address while BIU 20 initiates an external bus cycle to read the data from external memory. If the reference is a hit, then the selected data is aligned by Data Cache 16 and transferred to Execution Unit 32 on the GDATA bus and BIU 20 cancels the external bus cycle but does not assert the BMT and $\overline{CONF}$ signals. If the reference is a

miss, BIU 20 completes the external bus cycle and transfers data from external memory to Execution Unit 32 and to Data Cache 16, which updates its cache entry. For references that hit, Data Cache 16 can sustain a throughput of one double-word per cycle, with a latency of 1.5 cycles.

Data Cache 16 is a write-through cache. For memory write references, Data Cache 16 examines whether the reference is a hit. If so, the contents of the cache are updated. In the event of either a hit or a miss, BIU 20 writes the data through to external memory.

Like Instruction Cache 14, Data Cache 16 supports an operating mode to lock its contents to fixed locations. This feature is enabled by setting the Lock Data Cache (LDC) bit in the configuration register. It can be used in real-time systems to allow fast on-chip access to the most critical data locations. Data Cache 16 can be enabled by setting the Data Cache Enable (DC) bit in the configuration register.

The configuration register included in Register File 34 is configured in 32 bits, of which 9 bits are implemented. The implemented bits enable various operating modes for CPU 10, including vectoring of interrupts, execution of slave instructions, and control of the on-chip Instruction Cache 14 and Data Cache 16. When the contents of the configuration register are loaded, the values loaded to bits 4 through 7 are ignored; when the contents of the configuration register are stored, these bits are 1.

The format of the configuration register is shown in Table 5. The various control bits are described below.

| x | x | x | LIC | IC | LDC | DC | DE | 1 | 1 | 1 | 1 | C | M | F | I |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | | | | | | | | | | | | | | | 0 |

Table 5

I Interrupt vectoring. This bit controls whether maskable interrupts are handled in nonvectored (VI=0) or vectored (VI=1) mode.

F Floating-point instruction set. This bit indicates whether a floating-point unit is present to execute floating-point instructions.

M Memory management instruction set. This bit enables the execution of memory management instructions.

C Custom instruction set. This bit indicates whether a custom slave processor is present to execute custom instructions.

DE Direct-Exception enable. This bit enables a Direct-Exception mode, a mode of processing exceptions that improves response time of CPU 10 to interrupts and other exceptions.

DC Data Cache Enable. This bit enables Data Cache 16 to be accessed for data reads and writes.

LDC Lock Data Cache. This bit controls whether the contents of Data Cache 16 are located to

fixed memory locations (LDC=1) or updated when a data read is missing from the cache (LIC=0).

IC Instruction Cache Enable. This bit enables Instruction Cache 14 to be accessed for instruction fetches.

LIC Lock Instruction Cache. This bit controls whether the contents of Instruction Cache 14 are located to fixed memory locations (LIC=1) or updated when an instruction fetch is missing from the cache (LIC=0).

As stated above, CPU 10 overlaps operations to execute several instructions simultaneously in 4-stage Pipeline 12. The general structure of Pipeline 12 and the various buffers for instructions and data are shown in Fig. 7. While Execution Unit 32 is calculating the results for one instruction, Address Unit 30 can be calculating the effective addresses and reading the source operands for the following instruction, and Loader 28 can be decoding a third instruction and prefetching a fourth instruction into its 8-byte queue.

Address Unit 30 and Execution Unit 32 can process instructions at a peak rate of two cycles per instruction. Loader 28 can process instructions at a peak rate of one cycle per instruction, so it will typically maintain a steady supply of instructions to Address Unit 30 and Execution Unit 32. Loader 28 disrupts the throughput of Pipeline 12 only when a gap in the instruction stream arises due to a branch instruction or a miss in Instruction Cache 14.

Fig. 8 shows the execution of two memory-to-register instructions by Address Unit 30 and Execution

Unit 32. CPU 10 can sustain an execution rate of two cycles for most common instruction, typically exhibiting delays only in the following cases:

1. Storage delays due to cache and translation buffer misses and non-aligned references.
2. Resource contention between stages of Pipeline 12.
3. Branch instruction and other non-sequential instruction fetches.
4. Complex addressing modes like scaled index, and complex operations, like division.

Fig. 9 shows the effect of a miss in Data Cache 16 on the timing of Pipeline 12. Execution Unit 32 is delayed by two cycles until BIU 20 completes the bus cycles to read data. The basic bus cycles performed by CPU 10 are discussed in greater detail below.

Fig. 10 shows the effect of an address-register interlock on the timing of Pipeline 12. One instruction is modifying a register while the next instruction uses that register for an address calculation. Address Unit 30 is delayed by three cycles until Execution Unit 32 completes the register's update. Note that if the second instruction had used the register for a data value rather than an address calculation (e.g., ADDD R0, R1), then bypass circuitry in Execution Unit 32 would be used to avoid any delay to Pipeline 12.

As stated above, Loader 28 includes circuitry for the handling of branch instructions.

"Branch" instructions are those instructions that potentially transfer control to an instruction at a destination address calculated by adding a displacement value encoded into the currently executing instruction to the address of the currently executing instruction. Branch instructions can be "unconditional" or

"conditional"; in the latter case, a test is made to determine whether a specified condition concerning the state of CPU 10 is true. A branch instruction is said to be "taken" either if it is unconditional or if it is conditional and the specified condition is true.

When a branch instruction is decoded, Loader 28 calculates the destination address and selects between the sequential and non-sequential instruction streams. The selection is based on the branch instruction condition and direction. If Loader 28 predicts that the branch instruction is taken, then the destination address is transferred to Instruction Cache 14 on the JBUS. Whether or not the branch instruction is predicted to be taken, Loader 28 saves the address of the alternate instruction stream. Later the branch instruction reaches Execution Unit 32, where the condition is resolved. Execution Unit 32 signals Loader 28 whether or not the branch instruction was taken. If the branch instruction had been incorrectly predicted, Pipeline 12 is flushed and Instruction Cache 14 begins prefetching instructions from the correct stream.

Fig. 11 shows the effect of correctly predicting a branch instruction to be taken. A 2-cycle gap occurs in the decoding of instructions by Loader 28. This gap at the very top of Pipeline 12 can often be closed because one fully decoded instruction is buffered between Loader 28 and Address Unit 30 and because other delays may arise simultaneously at later stages in Pipeline 12.

Fig. 12 shows the effect of incorrectly predicting the resolution of a branch instruction. A 4-cycle gap occurs at Execution Unit 32.

CPU 10 receives a single-phase input clock CLK which has a frequency twice that of the operating rate

of CPU 10. For example, the input clock's frequency is 40 MHz for a CPU 10 operating at 20 MHz. CPU 10 divides the CLK input by two to obtain an internal clock that is composed of two non-overlapping phases, PHI1 and PHI2. CPU 10 drives PHI1 on the BUSCLK output signal.

Fig. 13 shows the relationship between the CLK input and BUSCLK output signals.

As illustrated in Fig. 14, every rising edge of the BUSCLK output defines a transition in the timing state ("T-state") of CPU 10. Bus cycles occur during a sequence of T-states, labelled T1, T2, and T2B in the associated timing diagrams. There may be idle T-states (Ti) between bus cycles. The phase relationship of the BUSCLK output to the CLK input can be established at reset.

The basic bus cycles performed by CPU 10 to read from and write to external main memory and peripheral devices occur during two cycles of the bus clock, called T1 and T2. The basic bus cycles can be extended beyond two clock cycles for two reasons. First, additional T2 cycles can be added to wait for slow memory and peripheral devices. Second, when reading from external memory, burst cycles (called "T2B") can be used to transfer multiple double-words from consecutive locations.

The timing for basic read and write bus cycles with no wait states is shown in Figs. 14 and 15, respectively. For both read and write bus cycles, CPU 10 asserts Address Strobe $\overline{\text{ADS}}$ during the first half of T1 indicating the beginning of the bus cycle. From the beginning of T1 until the completion of the bus cycle, CPU 10 drives the Address Bus and control signals for the Status (ST0-ST4), Byte Enables (BE0-BE3), Data Direction In ($\overline{\text{DDIN}}$), Cache Inhibit (CIO), I/O Inhibit

($\overline{\text{IOINH}}$), and Confirm Bus Cycle ($\overline{\text{CONF}}$) signals.

If the bus cycle is not cancelled (that is, T2 will follow on the next clock), CPU 10 asserts Begin Memory Transaction $\overline{\text{BMT}}$ during T1 and asserts Confirm Bus Cycle $\overline{\text{CONF}}$ from the middle of T1 until the completion of the bus cycle, at which time $\overline{\text{CONF}}$ is negated.

At the end of T2, CPU 10 samples whether RDY is active, indicating that the bus cycle has been completed; that is, no additional T2 states should be added. Following T2 is either T1 for the next bus cycle or Ti, if CPU 10 has no bus cycles to perform.

As shown in Fig. 16, the basic read and write bus cycles just described can be extended to support longer access times. As stated, CPU 10 samples RDY at the end of each T2 state. If RDY is inactive, then the bus cycle is extended by repeating T2 for another clock. The additional T2 states after the first are called "wait" states. Fig. 16 shows the extension of a read bus cycle with the addition of two wait states.

As shown in Fig. 17, the basic read cycles can also be extended to support burst transfers of up to four double-words from consecutive memory locations. During a burst read cycle, the initial double-word is transferred during a sequence of T1 and T2 states, like a basic read cycle. Subsequent double-words are transferred during states called "T2B". Burst cycles are used only to read from 32-bit wide memories.

The number of transfers in a burst read cycle is controlled by a handshake between output signal $\overline{\text{BREQ}}$ and input signal $\overline{\text{BACK}}$ during a T2 or T2B state to indicate that it requests another transfer following a current one. The memory asserts $\overline{\text{BACK}}$ to indicate that it can support another transfer. Fig. 17 shows a burst read cycle of three transfers in which CPU 10 terminates

the sequence by negating $\overline{BREQ}$ after the second transfer. Fig. 18 shows a burst cycle of two transfers terminated by the system when $\overline{BACK}$ was inactive during the second transfer.

For each transfer after the first in the burst sequence, CPU 10 increments address bits 2 and 3 to select the next double-word. As shown for the second transfer in Fig. 18, CPU 10 samples RDY at the end of each state T2B and extends the access time for the burst transfer if RDY is inactive..

High-speed address translation is performed on-chip by the above-referenced translation buffer which holds address mappings for 64 pages. The page size is 4K bytes. The translation buffer provides direct virtual to physical address mapping for recently-used memory pages. Entries in the translation buffer are allocated and replaced automatically by MMU 18. If the information necessary to translate a virtual address to a physical address is missing from the translation buffer, CPU 10 automatically locates the information from two levels of page table entries in external memory and updates the translation buffer. If MMU 18 detects a protection violation or page fault while translating an address for a reference required to execute an instruction, an abort trap occurs and the instruction being executed is suspended.

Each of the 64 entries in the translation buffer stores the virtual and physical page frame numbers, i.e., the 20 most-significant bits of the address, along with the address space for the virtual page, the protection level for the page, and modified and cache inhibit bits from the level-2 page table entry.

The protection level field determines the protection level assigned to a certain page or group of

pages. Table 6 shows the encoding of the protection level field.

Table 6

| Address Space | AS | Protection - Level Field 00 | 01 | 10 | 11 |
|---|---|---|---|---|---|
| User | 1 | no access | no access | read access | full access |
| Supervisor | 0 | read only | full access | full access | full access |

As stated above, a cache inhibit bit CI appears in second-level page table entries. If the cache inhibit bit is 1, then instruction-fetch and data-read references to locations on the page by-pass the on-chip caches. The cache inhibit bit is indicated on the system interface during references to external memory.

The modified bit also appears in second-level page table entries. MMU 18 sets the modified bit in the page table entry to 1 whenever a write is performed to the page and the modified bit in the page table entry is 0.

To translate a virtual address to the corresponding physical address, the virtual page frame number and the address space are compared with the entries in the translation buffer. If a valid entry with a matching page frame number and address space is already present in the translation buffer, the physical address is available immediately. Otherwise, if no valid entry in the translation buffer has the matching page frame number and address space, MMU 18 translates the virtual address and places the missing information into the translation buffer. MMU 18 also performs a translation upon writing to a page that has not been previously modified.

When translation is enabled for a memory reference, MMU 18 translates 32-bit virtual addresses to 32-bit

physical addresses, checking for protection violations on each reference and possibly inhibiting the use of the on-chip cache for the reference, as described above. When translation is disabled for a reference, the physical address is identical to the virtual address, no protection checking is performed and the on-chip caches are not inhibited for the reference.

As stated above, MMU 18 translates addresses using 4KB pages and two levels of translation tables. The virtual address is divided into three components: INDEX1, INDEX2 and OFFSET. INDEX1 and INDEX2 are both 10-bit fields used to point into the first and second level page tables, respectively. OFFSET is the lower 12 bits of the virtual address; it points to a byte within the selected page.

When reading page table entries during address translation, MMU 18 bypasses Data Cache 16, referring always to external memory. When updating a page table entry that is located in Data Cache 16, MMU 18 updates the contents of the page table entry both in Data Cache 16 and in external memory.

The system interface of CPU 10 also supports the use of external cache memory 25, as shown in Figure 1. The CI bit from the level-2 page table entries is presented on the CIO output signal during a bus cycle along with the address, allowing individual pages to be selectively cached. CPU 10 can also be made to retry a bus cycle by asserting the $\overline{\text{BRT}}$ input signal during the bus cycle. Before trying the bus cycle again, CPU 10 releases the bus, thereby allowing external cache 25 to handle misses by performing accesses to external main memory.

In accordance with the present invention, CPU 10 provides for maintaining coherence between the two on-

chip caches and external memory. The techniques utilized by CPU 10 for this purpose are summarized in Table 7.

| | SOFTWARE | HARDWARE |
|---|---|---|
| Inhibit Cache Access for certain locations | Cache-Inhibit CI bit in PTE | Cache-Inhibit input signal |
| Invalidate certain locations in Cache | CINV Instruction to invalidate block | Cache Invalidation request to invalidate set |
| Invalidate Entire Cache | CINV Instruction | Cache Invalidation request |

Table 7

As stated above, the use of caches can be inhibited for individual pages using the CI bit in the level-2 page table entries.

Entries in Instruction Cache 14 and Data Cache 16 can be invalidated using the Cache Invalidate CINV instruction. While executing the CINV instruction, CPU 10 generates two slave bus cycles on the system interface to display the first 3 bytes of the instruction and the source operand. External circuitry can thereby detect the execution of the CINV instruction for use in monitoring the contents of the on-chip caches.

The CINV instruction can be used to invalidate either the entire contents of either or both of the internal caches or only a 16-byte block in a selected cache. In the latter case, the 28 most significant bits of the source operand specify the physical address of the aligned 16-byte block; the 4 least significant bits of the source operand are ignored. If the specified block is not located in the on-chip caches, then the instruction has no effect. The CINV instruction refers

to Instruction Cache 14 according to an I-opti n and to Data Cache 16 according to a D-option.

The format of the CINV instruction is shown in Table 8.

| SRC | OPTIONS | | | | CINV |
|---|---|---|---|---|---|
| GEN | 0 | A | I | D | 0 1 0 0 1 1 1 0 0 0 1 1 1 1 0 |
| 23 | | 16 | | | 8 7 0 |

Table 8.

Options are specified by listing the letters A (invalidate all), I (Instruction Cache) and D (Data Cache). If neither I nor D options are specified, nothing is invalidated.

In the machine instruction, the options are encoded in the A, I and D fields as follows:

A: 0 = invalidate only a 16-byte block
1 = invalidate the entire cache

I: 0 = do not affect the Instruction Cache
1 = invalidate the Instruction Cache

D: 0 = do not affect the Data Cache
1 = invalidate the Data Cache

CPU 10 also supports an external "Bus-Watcher" circuit 26, shown in Fig. 1. The primary function of Bus Watcher 26 is to maintain coherence between Instruction Cache 14 and Data Cache 16 on one hand and external memory on the other hand in either a shared-memory multiprocessor system or a single-processor system with high-bandwidth direct memory access DMA devices. Bus-Watcher 26 observes the bus cycles of CPU 10 to maintain a copy of the cache address tags for Instruction Cache 14 and Data Cache 16 while also monitoring writes to external memory by, for example, DMA controllers and other microprocessors in the system. When Bus-Watcher 26 detects, through a comparison of the cache address tags and the write reference address, that

a location in the on-chip cache has been modified in external memory, it signals a cache invalidation request to CPU 10.

As shown in Fig. 19, Bus-Watcher 26 interfaces to the following buses:

1. CPU 10 Address Bus and CASEC output, to get information on which internal cache entries (tags) are modified and to maintain updated copies of CPU 10 internal cache tags;

2. The System Bus, to detect which external memory addresses are modified; and

3. CPU 10 Cache Invalidation bus, consisting of the $\overline{\text{INVSET}}$, $\overline{\text{INVDC}}$, $\overline{\text{INVIC}}$ and CIA0-CIA6 input signals.

Referring to Fig. 19, and as stated above, Bus-Watcher 26 maintains tag copies of the two internal caches of CPU 10, Instruction Cache 14 and Data Cache 18. If the address of a memory write cycle on the System Bus matches one of the tags inside Bus-Watcher 26, a command is issued by Bus-Watcher 26 to CPU 10, via the Cache Invalidation Bus, to invalidate the corresponding entry in the internal cache. Since the invalidation signal is provided over the separate Cache Invalidation Bus, the invalidation of the internal cache entry by CPU 10 takes one clock cycle only and does not interfere with an on-going external bus cycle of CPU 10. Instruction Cache 14 and Data Cache 16 are invalidated one set at a time; i.e. 16 bytes in Instruction Cache 14 and 32 bytes in Data Cache 16.

The input signal INVSET indicates whether the invalidation applies to a single set (low) or to the entire cache (high).

If the invalidation request occurs prior to or at the same time that CPU 10 is completing a T2 or T2B

state in a read cycle to a location affected by the invalidation, the data read on the bus will be valid in the cache. If the invalidation request occurs after the T2 or T2B state in the read cycle, the data will be invalid in the cache.

When the Invalidate Instruction Cache INVIC input is low, the invalidation is done in Instruction Cache 14.

When the Invalidate Data Cache INVDC input is low, the invalidation is done in Data Cache 16.

The Cache Invalidation Address CIA0-CIA6 is presented to CPU 10 on the CIA bus. The bits provide the set address to be invalidated in Data Cache 16 and in Instruction Cache 14.

The Bus Watcher circuitry consists primarily of three RAM arrays that store the copies of the cache address tags. The RAM bits are, as shown in Fig. 19, dual ported. One port is used for writing the tags during bus read cycles by CPU 10. The second port is used for reading the tags during invalidation requests from the system bus. By using dual-ported memory cells, problems associated with synchronizing the system bus invalidation requests with the bus cycles of CPU 10 are simplified.

In addition to avoiding interference with the external references of CPU 10, use of Bus-Watcher 26 also avoids interference with the internal activity of CPU 10. This is accomplished through the use of dual-ported validity bits in both Instruction Cache 14 and Data Cache 16, as described above in relation to Figs. 4 and 5.

The system requirements for utilization of Bus Watcher 26 depend on the rate of potential cache invalidations caused by modifications of shared memory.

The cache invalidation mechanisms of CPU 10, i. ., the CINV instruction described above, can be used without Bus Watcher 26 if the rate of potential invalidations is much lower than the rate of memory accesses by CPU 10. For example, systems that implement write-through policies cause a high rate of potential invalidations and would require Bus Watcher 26; systems that use write-back policies may have a rate of potential invalidations sufficiently low that Bus Watcher 26 is unnecessary.

Three possible internal cache invalidation scenarios are illustrated in Figs. 20-22.

Fig. 20 shows a cache coherence solution for a system that requires a low invalidation rate and, therefore, does not utilize Bus-Watcher 26. When a DMA controller or another CPU in the system writes to the contents of Location A in main memory on the system bus, the seven lower bits of the address of Location A are provided to CPU 10 on the CIA bus and both the INVIC and INVDC inputs are driven low such that the set which includes Location A on-chip is invalidated. That is, without the screening provided by Bus-Watcher 26, any write on the system bus will invalidate a set in Instruction Cache 14 and Data Cache 16. This design is applicable to uniprocessor systems or certain types of multiprocessor organizations, e.g. those that use ownership schemes for memory.

Fig. 21 shows a cache coherence solution for a system which must support a high cache invalidation rate and, thus, warrants use of Bus-Watcher 26. As stated above, Bus-Watcher 26 maintains a copy of the on-chip cache tags. Thus, any write on the system bus which produces a match with a Bus-Watcher tag will cause an

invalidation in the corresponding internal cache based on the CIA, INVDC, INVIC and INVSET inputs.

A third cache coherence solution is shown in Fig. 22. This system has a high invalidation rate but also incorporates a large external cache 25. In this case, external cache 25 maintains coherence with main memory by using its own bus watcher. The internal caches, therefore, needs only to maintain coherence with external cache 25. Any invalidation to external cache 25 invalidates a set in the internal cache. Any update to external cache 25 invalidates a set in the internal cache.

Additional information regarding the operation of CPU 10 may be found in copending and commonly-assigned U.S. Pat. Appln. Serial No. 006,016, "High Performance Microprocessor", filed by Alpert et al of even date herewith, and which is hereby incorporated by reference.

What is claimed is:

1. A method of maintaining coherence between an integrated cache memory of a microprocessor and the microprocessor's associated external memory, wherein the microprocessor communicates with the external memory via an external data bus, the method comprising executing a cache invalidate instruction which invalidates information stored in the integrated cache memory.

2. A method as in claim 1 wherein the integrated cache memory comprises an instruction cache and separate data cache.

3. A method as in claim 2 wherein the cache invalidation instruction invalidates the entire contents of both the instruction cache and the data cache.

4. A method as in claim 2 wherein the cache invalidation instruction invalidates the entire contents of the instruction cache.

5. A method as in claim 2 wherein the cache invalidation instruction invalidates specified contents of the instruction cache.

6. A method as in claim 2 wherein the cache invalidation instruction invalidates the entire contents of the data cache.

7. A method as in claim 2 wherein the cache invalidation instruction invalidates specified contents of the data cache.

8. A m thod as in claim 2 wherein the cache invalidation instruction invalidates specified contents of both the instruction cache and the data cache simultaneously.

9. A system for maintaining coherence between an integrated cache memory of a microprocessor and the microprocessor's associated external memory, wherein the microprocessor communicates with the external memory via an external data bus and wherein the external data bus is used by devices external to the microprocessor to modify the information stored in the external memory, the system comprising:

storage means for maintaining address tags corresponding to addresses of information stored in the integrated cache memory;

means for monitoring the external data bus to identify addresses of writes to the external memory by the external devices;

means for comparing a write address with the stored address tags to detect a match between the write address and an address of information stored in the integrated cache memory; and

means for generating a request to the microprocessor to invalidate information stored in the integrated cache memory in response to a match between the write address and an address of information stored in the integrated cache memory.

10. A system as in claim 9 and further including a cache invalidation bus, separate from the external data bus, which transfer the cache invalidation request to the microprocessor.

11. A system as in claim 10 wherein the cache invalidation request specifies the address of the location to be invalidated within the integrated cache memory.

12. A system as in claim 10 wherein the cache invalidation request specifies that all information stored in the integrated cache memory is to be invalidated.

13. A system as in claim 10 wherein the integrated cache memory comprises an instruction cache and a separate data cache.

14. A system as in claim 13 wherein the cache invalidation requests specifies that all information stored in both the instruction cache and the data cache is to be invalidated.

15. A system as in claim 13 wherein the cache invalidation request specifies that all information stored in the instruction cache is to be invalidated.

16. A system as in claim 13 wherein the cache invalidation request specifies the address of the location to be invalidated within the instruction cache.

17. A system as in claim 13 wherein the cache invalidation request specifies that all information stored in the data cache is to be invalidated.

18. A system as in claim 13 wherein the cache invalidation request specifies the address of the location to be invalidated within the data cache.

19. A system as in claim 13 wherein the cache invalidation request specifies the address of the location to be simultaneously invalidated within both the instruction cache and the data cache.

20. A system as in claim 13 wherein the data cache comprises a plurality of sets and the cache invalidation request specifies the set to be invalidated.

21. A method of maintaining coherence between an integrated cache memory of a microprocessor and the microprocessor's external memory, wherein the microprocessor communicates with the external memory via an external data bus and wherein the external data bus is used by devices external to the microprocessor to modify information stored in the external memory, the method comprising:

maintaining address tags corresponding to address of information stored in the integrated cache memory;

monitoring the external data bus to identify addresses of writes to the external memory by the external devices;

comparing a write address with the stored address tags to detect a match between the write address and an address of information stored in the integrated cache memory;

in response to a match between the write address and an address of information stored in the integrated cache memory, generating a request to the microprocessor to invalidate information stored in the integrated cache memory.

22. A method as in Claim 21 wherein the cache invalidation request is provided to the microprocessor by a cache invalidation bus separate from the external data bus.

23. A method as in Claim 22 wherein the cache invalidation request specifies a location to be invalidated within the integrated cache memory.

24. A method as in Claim 23 including the further step of externally monitoring the location invalidated within the integrated cache memory.

25. A method of maintaining coherence between an integrated cache memory of a microprocessor and the microprocessor's associated external memory as claimed in Claim 1 or Claim 21 substantially as herein described with reference to the accompanying drawings.

26. A system for maintaining coherence between an integrated cache memory of a microprocessor and the microprocessor's associated external memory as claimed in Claim 9 substantially as herein described with reference to the accompanying drawings.

Published 1988 at The Patent Office, State House, 66/71 High Holborn, London WC1R 4TP. Further copies may be obtained from The Patent Office, Sales Branch, St Mary Cray, Orpington, Kent BR5 3RD. Printed by Multiplex techniques ltd, St Mary Cray, Kent. Con. 1/87.